# CSC460 Theory of Computation (Spring 2005)

## Instructor

- Nobo Komagata, komagata@tcnj.edu, Holman Hall 229, Office hours: See http://www.tcnj.edu/~komagata/schedule

## On-line Resources (required reading)

- Course web page: http://www.tcnj.edu/~komagata/csc460/05s
- The instructor's information page for students (http://www.tcnj.edu/~komagata/students.html)
- SOCS: http://socs.tcnj.edu/ (the following sections will be used for this course)
  - E-mail: Convenient to send messages to the class or a group of students.
  - Discussion: The SOCS discussion board will be used for certain types of announcements, question-answer, and discussion. In order to get informed of postings in a timely manner, students should **turn on the *Email Notification* feature** of the SOCS discussion board.
  - Grading: Used for posting module and course grades. Note that the "Grade" field will show "1" when the grade is available and the actual grade will appear in the "Note" field.

## Catalog Course Description

This course focuses on the traditional, algorithmic theory of computation consisting of three subareas: (1) computability, (2) complexity theory, and (3) formal languages and automata. The topics include: Turing machines, decidability/undecidability, reducibility, Church-Turing thesis, context-free grammars/languages, push-down automata, finite automata, regular expressions/languages, and time/space complexity including NP-completeness. [Prerequisites: CSC310, CSC340]

## Learning Goals

In this course, we examine a small number of general principles that lie behind a variety of computational phenomena so that we will be able to apply these principles to a broad range of real-world computation. One of the main goals of this course is to convince students that "theory" is useful for practically anyone. To do so, we need to observe examples of theory at work. Once we understand the benefits of using theory in general, we will be ready to explore Theory of Computation, which has always been behind the development of computer science. At the same time, we will also examine the limitations of Theory and identify what is really behind real-world computation. In this course, we will discuss most of the topics traditionally covered in an undergraduate-level Theory of Computation course. However, our approach will be to explore the possibility of applying Theory to practical problems which students are interested in. The course will not follow the textbook, nor will it do most of the textbook problems.

***Content Goals*** (mastery of the following core concepts, deep understandings, misunderstandings, and technical knowledge)

1. Practical problems can often be transformed into research and computational problems. Every problem is associated with cost/significance, which is relative to the evaluator. Computational problems can be represented as a set, readily available as the input to computational processing. [problem]
2. A theory is a potentially infinite, consistent body of knowledge which can be systematically derived from a small number of abstract principles. The gap between the principles and the entire information is the source of the theory's predictive power. Also being abstract, a theory can be applied to a broad range of phenomena, which might appear distinct. [theory]
3. Interactive computation subsumes algorithmic computation, but not vice versa. That is, there is a qualitative difference between these two modes of computation. [interactive computation]
4. The algorithmic notion of computation can be represented in a variety of equivalent forms, which define a bounded class of sets. That is, there is a limit to algorithmic computation. [computability]
5. The computable class of sets contain a hierarchy of proper subsets which can be characterized by distinct grammars and automata. [formal languages and automata]
6. The practicality of an algorithm depends on its complexity relative to the input data size. [complexity]
7. Power set, also encompassing the distinction between determinism and nondeterminism, can introduce discontinuity with respect to computability and complexity. [power set]

***Performance Goals*** (expected outcomes and abilities to be observed as a result of successful learning)

1. Identify real-world problems which are relevant to the student's life and can be tackled by computational means. [awareness]
2. Transform real-world problems into research problems, and then into computational problems, along with the analysis of the cost/significance of a problem. [transformation]
3. Analyze computational problems with respect to interactivity, computability, language/automata hierarchy, and complexity hierarchy. Then, evaluate the analysis with respect to its usefulness, correctness, and accuracy. [analysis/evaluation]
4. Respect, analyze, and give constructive criticisms to the ideas in the literature and those expressed by other people. [critical attitude]
5. Express ideas orally and in writing, in a manner clearly understood by other students (with equivalent background). Explain your own ideas orally and in writing, clearly and logically. Revise the ideas, reflecting the feedback from other students and the instructor. [communication]
6. Take initiative in both independent and group activities. Also extend the domain of theoretical inquiry beyond the scope prepared by the instructor. [initiative]
7. Reflect upon the student's own thinking process and assess the student's own performance relative to the content and performance goals. [reflection]

## Course Modules

This course is divided into the following four modules.

- Module A: Problems, Theories, and Computability (1)
- Module B: Computability (2)
- Module C: Formal Languages and Automata
- Module D: Complexity

However, the topics of this course cannot be clearly separated into disjoint components. Thus, these modules should not be considered rigidly. Instead, these modules should be considered as evaluation units so that students can check their achievements in a timely manner.

## Assessment

This course is expected to facilitate learning experience with which students can analyze practical computational problems in a intellectual manner by applying principles in Theory of Computation. Therefore, students' assessment must reflect their ability to do that in a realistic context. Take-home exercises are designed to provide activities reflecting the learning goals (except for Performance Goal 8) in a realistic context. Thus, students' assessment will primarily based on how well they accomplish these exercises. As for Performance Goal 8, evaluation workshops will provide opportunities for students to reflect upon their thought processes and self-evaluate their achievements with respect to the learning goals.

The main assessment tools are students' self-evaluation at the end of each module. Students will evaluate themselves with respect to the learning goals following the instructions on the self-evaluation form (preliminary eval form for Module A). Since students will be given the form at the beginning of each module, they will be aware of what they are expected to achieve for that module. The evaluation form will list the learning goals relevant to the module as well as evaluation criteria for the relevant goals. Analysis and reflections on take-home exercises will be most important as the exercises address the learning goals directly or indirectly. Students are also expected to respond to the instructor's feedback on their take-home exercises. The evaluation workshop will generally include peer discussion sessions, when students can share their learning and evaluation experiences.

The grading scheme for self-evaluation (and the instructor's adjustment) is as follows:

- Grade A: Achieved all the learning goals relevant to the module
- Grade B: Achieved almost all the learning goals (except for one or two evaluation criteria) relevant to the module
- Grade C/Pass: Achieved most of the learning goals relevant to the module

The grades A, B, and C may be qualified with '+' or '–', where applicable. The overall course grade will be the weighted average of the module grades as shown below (using the standard conversion A = 4.00, A– = 3.67, etc., according to http://www.tcnj.edu/~recreg/policies/grading.html#three, *not* http://www.tcnj.edu/~academic/policy/Grading.htm).

- Module A: 10% (lower weighting for practice purposes)
- Module B: 30%
- Module C: 30%
- Module D: 30%

If a student far exceeds the standard, s/he may be assigned an A+. Although there will be no course grade beyond A, an A+ (for a module) could offset, say, A– in another module. For example, if a student receives A, A–, A, and A+ for the four modules, respectively, s/he will receive an A for the course.

After almost every class meeting, there will be take-home exercises. Students are expected to do them and hand them in at the beginning of the next class meeting. Although these exercises will not be graded by the instructor, the instructor will give feedback on them and return them at the beginning of the next class meeting. When students self-evaluate, they will submit all the take-home exercises along with their responses to the instructors' feedback. After submission of a module evaluation folder (called "portfolio"), the instructor will adjust students' self-evaluation, provide comments, and temporarily return them to students for review (when possible). Note that all the submitted work will eventually be kept with the instructor.

## Learning Activities

The main learning activities consist of class meetings (two 80-minute sessions per week) and take-home exercises (expected amount of work equivalent to 360 minutes per week). Take-home exercises will be given after almost every class meeting and due at the beginning of the next class meeting (unless otherwise stated). Class meetings will contain components such as the following:

- Clarification of the learning goals as well as explanation of the evaluation form
- Survey (occasionally)
- Presentation of examples, cases, questions, and problems by students and the instructor
- Discussion of ideas, principles, and hypotheses known by the public and researchers
- Class and group discussion of different types
- Explanation and practice of using the evaluation form (esp. during Module A)
- Explanation of the take-home exercises given that day
- Evaluation workshop (at the end of each module)

**Course Topics** (subject contents)

1. Types of problems
    1. Practical, research, and computational problems
    2. Condition and cost/significance of a problem
    3. Set representation of a problem
2. Notion of "theory"
    1. Applicability of a theory
    2. Definition in the context of mathematical logic
3. Computability
    1. Formal models of computation, e.g., Turing machines (TMs)
    2. Decidability/undecidability including diagonalization and halting problem
    3. Reduction
    4. Church-Turing thesis
4. Formal languages and automata
    1. Chomsky hierarchy
    2. Context-free languages (CFL), context-free grammars (CFG), push-down automata (PDA); properties of CFLs
    3. Regular languages, regular expressions (RegExp), finite automata (FA); properties of regular languages
5. Complexity
    1. Tractability/Intractability
    2. Nondeterministic polynomial
    3. NP-complete
    4. Space complexity
6. Interactivity
    1. Effects of interaction
    2. Formal models of interaction

## Textbook

- Hopcroft, John E., Motwani, Rajeev, and Ullman, Jeffrey D. 2001. *Introduction to Automata Theory, Languages, and Computation, 2nd ed*. Addison-Wesley. ISBN: 0-201-44124-1. [**required** (but listen to what the instructor will say about how we will use the textbook); available in the TCNJ bookstore; also available as 3-hour reserve in the library]

### Schedule: Class meetings: Tue/Fri 4:00-5:20 p.m. in HH126

| Week | Date | Unit | Topic | Exercises |
|---|---|---|---|---|
| 1 | 1/18 | 00 | Introduction | Your own computational problems |
| | 1/21 | A1 | Problems: Anatomy and physiology | Using a theory |
| 2 | 1/25 | A2 | Theory: Form and content | Your ideas about Theory of Computation |
| | 1/28 | A3 | Theory of Computation; Possibilities and limitations | Computing set membership |
| 3 | 2/1 | A4 | Turing Machines: Vehicle for mechanistic touring | Test drive Turing machines |
| | 2/4 | A5 | (Un)Decidability: When to play dice | **Module A Comprehensive Exercise** |
| 4 | 2/8 | A6 | **Module A Evaluation Workshop** [eval form] | Simulating a TM using a TM |
| | 2/11 | B1 | Computability: Universal TM | Universal TM review |
| 5 | 2/15 | B2 | Diagonalization: Magic of obliqueness | Diagonalize other things |
| | 2/18 | B3 | Halting Problem: Can we prove that we too will die? | Infinite loop detection |
| 6 | 2/22 | B4 | Reduction: How to get more from reduced stuff | Reduction; TM vs. Computers |
| | 2/25 | B5 | Church-Turing thesis: Civil rights movement in its abstract form | **Module B Comprehensive Exercise** |
| 7 | 3/1 | B6 | **Module B Evaluation Workshop** [eval form] | Are TM's too powerful or too powerless? |
| | 3/4 | B7 | Computability review | - |
| - | 3/8<br>3/11 | - | *No class (Spring break)* | - |
| 8 | 3/15 | C1 | Chomsky hierarchy: Often, we want less power | TM/grammar variants |
| | 3/18 | C2 | CFG, CFL, PDA: Shopping-mall navigation | CF examples |
| 9 | 3/22 | C3 | Regular expression, regular set, FA: Vending-machine operation | Regular examples |
| | 3/25 | C4 | Properties of regular sets: Pumping gas for free | Pumping exercise |
| 10 | 3/29 | C5 | Properties of CFLs: Pumping air into the lung; Chomsky hierarchy summary | **Module C Comprehensive Exercise** |
| | 4/1 | C6 | **Module C Evaluation Workshop** [eval form] | Data-size scalability |
| 11 | 4/5 | D1 | Complexity: Polynomial, Exponential, Nondeterminism | Complexity analysis |
| | 4/8 | D2 | NP-complete: Why SAT rules | Second civil rights movement |
| 12 | 4/12 | D3 | Space complexity: Is *time* the 4th dimension of *space*? | Limitations with complexity analysis |
| | 4/15 | D4 | Complexity review: What really matters | TBA |
| 13 | 4/19 | D5 | "Real" complexity (1): Effects of interaction | TBA |
| | 4/22 | D6 | "Real" complexity (2): Modeling interaction | **Module D Comprehensive Exercise** |
| 14 | 4/26 | D7 | D6: **Final Evaluation Workshop** [eval form]; Conclusion | - |
| Final | TBA | ZZ | Closing circle (optional; to be explained later) | - |

- The schedule is tentative and subject to change. Please always refer to the on-line syllabus, which contains the latest information.
- Course materials will be linked to the text in blue in the on-line syllabus.

// End

# CSC460 (Spring 2005) Theory of Computation
## List of Definitions, Notations, and Concepts

Jan. 14, 2005

Note: The section and page numbers refer to the text (Hopcroft et al., 2001). For other references, see the reference list at the end of this document.

## Problems

**Practical problem** [definition]: phenomenon (generally perceived negatively) that calls for some *action* [ref. Booth et al.]

**Research problems** [definition]: question (yes-no or *wh*-question) that is generally crucial to solve a practical problem and calls for an *answer* [ref. Booth et al.]

**Computational problems** [definition]: (informally) research problem that rends to computational some computational processing/analysis; (formally, within the traditional Theory) membership problem with respect to a certain set (thus, can be identified with the **set**); ~ language Sec 1.5.4

**Cost/significance** (of a problem) [definition]: The "cost" of a problem is what one would suffer if the problem is not solved. The "significance" of a problem is what one would benefit if the problem is solved. Thus, these two notions are the opposite side of the same idea (i.e., you only need to identify one, not both). While the process of solving a problem would be *objective* and can be purely technical ("value-free"), the cost/significance of the problem is *subjective* ("value-laden"). [ref. Booth et al.]

## Theory [ref. Enderton]

**Axioms** [definition]: the very basic principles consistent with our experience; the starting point of a theory (Note that there is no way to "logically" justify these; these are assumed and generally obtained intuitively.)

**Rules of inference** [definition]: the basic rules of obtaining additional information (statements) from the axioms

**Theorem** [definition]: statement justifiable from the axioms and the rules of inference

**Proof** [definition]: The "mechanical" process of justifying a theorem

**Logical consequence** [definition]: If a statement follows another by analyzing their "meanings," the former is called a "logical consequence" of the latter. (Note that the notion of "logical consequence" is stronger than that of "proof," and thus, there may be a logical consequence of the axioms that may not be proven.)

**Theory** [definition]: (informally) potentially infinite, consistent body of knowledge which can be systematically derived from a small number of abstract principles; (formally) the collection of all the statements that are logical consequences of the axioms

## Computability

**Turing machine** (**TM**) [definition] Sec 8.2

**Effective procedure** [description]: well-defined, step-by-step procedure Sec 8.2

**Algorithm** [description]: effective procedure that always terminates ~ recursive Sec. 8.2.6, 9.2.1, p. 367

(Computational) **Problem** [definition]: = set ~ property ~ predicate ~ characteristic function ~ language Sec 1.5.4

**Decidable** [definition]: = **recursive** = TM-decidable = computable = solvable Sec. 1.1.3, 8.2.6, 9.2.1, p. 374

**Undecidable** [definition] p. 302, 310, Ch 9

**TM-recognizable** [definition]: = **recursively enumerable** (**RE**) Sec 8.2.5, p. 374

**Unsolvable** [definition]: = **non-RE** = not TM-recognizable Sec 9.1, p. 374

**Semi-decidable** [definition]: = undecidable (but not unsolvable)

**co-*X*** (where *X* is some property) [definition]: The complement of the problem has property *X*. E.g., co-TM-recognizable. [ref. Sipser]

**Diagonalization** [description] Sec 9.1

**Rice's Theorem** Sec 9.3.3

**Church-Turing thesis** Sec 8.2.1

## Formal Languages and Automata

**Language** [definition]: set (often, of strings) Sec 1.5.4

**Finite-state automaton** (**FSA**; finite automaton, **FA**) [definition] Sec 2.1
  **Deterministic** finite-state automaton (**DFA**) [definition] Sec 2.2
  **Nondeterministic** finite-state automaton (**NFA**) [definition] Sec 2.3
**Regular expression** [definition] Sec 3.1
**Regular language** [definition] = **regular set** Sec 3.1
**Pumping lemma** [description] Sec 4.1
**Context-free grammar** (**CFG**) [definition] Sec 5.1
**Context-free language** (**CFL**) [definition] Sec 5.1
  **Deterministic** context-free language (**DCFL**) [definition] Sec 6.4
**Push-down automaton** (**PDA**) [definition] Sec 6.1
  **Deterministic** push-down automaton (**DPDA**) [definition] Sec 6.4
**Pumping lemma for CFLs** [description] Sec 7.2

## Complexity

**Complexity** (within Theory of Computation) [description]: the quantitative effects of the input size on computational performance
  **Time** complexity [description] p. 414
  **Space** complexity [description]: performance measure based on space (~ memory size) [ref. Sipser 1997]
  **Worst-case analysis** [description]: the performance analysis for the worst input pattern
  **Average-case analysis** [description]: the (estimated) average performance for all possible input patterns
  ‘***O***’ (asymptotic upper bound) [description] [ref. Sipser 1997]
    Informal idea: Behaves roughly as bad as a certain function
    Definition #1: $f(n) \in O(g(n))$ if there are constants $c > 0$ and $n_0 \geq 1$ such that $f(n) \leq c\, g(n)$ for every integer $n \geq n_0$.
    Definition #2: $f(n) \in O(g(n))$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$ for some $c \geq 0$

**Class P** (**tractable**) [definition] p. 361, Sec 10.1
**Class Exp** (EXPTIME) [description]: $O(c^n)$ for any constant $c > 1$
**Intractable** [definition] p. 1, p. 361, Sec 10.1
**Class NP** [definition] Sec 10.1
**Class NP-complete** [definition] Sec 10.2
**Cook-Levin theorem** [description] Sec 10.2.3
**Class NP-hard** [definition] p. 423

## General

**Power set** (of a set) [definition]: given set $A$, the set of all the subsets of $A$, i.e., $\{x \mid x \subseteq A\}$
**Deterministic** [definition]: at most one transition/option for a unique situation
**Nondeterministic** [definition]: possibly multiple transitions/options for a unique situation
**Reduction**, **reducibility** [definition] Sec 8.1.3
  **Polynomial** time reducibility [definition] Sec 10.1.5

## References [all available in the library]

Booth, Wayne C., Colomb, Gregory G., and Williams, Joseph M. 2003. *The craft of research*, 2nd ed. University of Chicago press. [excellent book on writing a research paper]
Enderton, Herbert B. 2001. *A mathematical introduction to logic*, 2nd ed. San Diego, CA: Harcourt. [classic introduction; only the 1st edition (1972) is available in the library]
Hopcroft, John E., Motwani, Rajeev, and Ullman, Jeffrey D. 2001. *Introduction to Automata Theory, Languages, and Computation*, 2nd ed. Addison-Wesley.
Hopcroft, John E. and Ullman, Jeffrey D. 1979 Introduction to automata theory, languages, and computation. Addison-Wesley. [the first edition of our text; contains some more advanced materials]
Sipser, Michael. 1997. *Introduction to the theory of computation*. PWS. [used as a textbook in the past]

// End

# **CSC460 References** (Spring 2005)

Apr. 20, 2005

The books available in the TCNJ library are indicated with "**L**".


## Standard texts in the Theory of Computation

- Greenlaw, Raymond and Hoover, H. James. 1998. *Fundamentals of the Theory of Computation*. Morgan Kaufmann. [another gentler introduction; little computability theory]
- Hein, James L. 1996. *Theory of Computation: An Introduction*. Jones and Bartlett.
- Hopcroft, John E., Motwani, Rajeev, and Ullman, Jeffrey D. 2001. *Introduction to Automata Theory, Languages, and Computation*, 2nd ed. Addison-Wesley. [the textbook; **L** (3-hour reserve)]
- Hopcroft, John E. and Ullman, Jeffrey D. 1979 Introduction to automata theory, languages, and computation. Addison-Wesley. [the first edition of our text; contains some more advanced materials; the discussion of the Chomsky hierarchy (Unit C1) is mainly based on this book; **L** (3-hour reserve)]
- Lewis, Harry R. and Papadimitriou, Christos H. 1981. *Elements of the Theory of Computation*. Prentice-Hall. [another classic; **L**]
- Linz, Peter. 2001. *An Introduction to Formal Languages and Automata*, 3rd ed. Jones & Bartlett. [very good, but little coverage of complexity theory; the discussion of the pumping lemma (Unit C3/C4) is mainly based on this book]
- Martin, John C. 1997. *Introduction to Languages and the Theory of Computation*, 2nd ed. WCB McGraw-Hill. [another undergrad text; **L**]
- Sipser, Michael. 1997. *Introduction to the theory of computation*. PWS. [used as a textbook in the past; space complexity ideas (Unit D3) are mainly taken from this book; **L** (3-hour reserve)]
- Sudkamp, Thomas A. 1997. *Languages and Machines*, 2nd ed. Addison Wesley.

## Other Theory of Computation references

- Atallah, Mikhail J, ed. 1999. *Algorithms and Theory of Computation Handbook*. CRC Press. [**L**]
- Baase, Sarah. 2000. *Computer Algorithms, Introduction to Design and Analysis*. Addison-Wesley. [classic text in algorithms; parallel algorithm examples (Unit D4) are mainly from this book]
- Blum, Lenore, et al. 1998. *Complexity and Real Computation*. Springer. [**L**]
- Cai, Jin-Yi. 2003. Lectures in Computational Complexity. [available on-line: http://www.cs.wisc.edu/~jyc/810notes/book-ch1to5.pdf]
- Chaitin, Gregory J. 1999. *The Unknowable*. Springer. [**L**]
- Cutland, N. J. 1980. *Computability: An Introduction to Recursive Function Theory*. Cambridge Univ. Press. [good intro to recursive function theory; recursive functions and URM ideas (Unit B5/B7) are mainly from this book]
- Davis, Martin. 1982. *Computability and Unsolvability*. Dover. [examples of unsolvable problems]
- Denning, Peter J., et al. 1978. *Machines, Languages, and Computation*. Prentice-Hall. [**L**]
- Dewdney, A. K. 1989. *The Turing Omnibus: 61 Excursions in Computer Science*. Computer Science Press. [a very readable introduction to various CS topics; the halting problem reading (Ex B1) and some Sample Problems are taken from this book; **L**]
- Epstein, Richard L. and Carnielli, Walter A. 1989. *Computability: Computable Functions, Logic, and the Foundations of Mathematics*. Wardsworth & Brooks/Cole. [interesting collection of topics with a lot of excerpts]
- Garey, Michael R. and Johnson, David S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman. [extensive collection of NP-complete problems; NPC problems and their connections (Unit D1/D2) are mainly taken from this book; **L**]
- Griffor, Er. R., ed. 1999. *Handbook of Computability Theory*. Elsevier. [**L**]
- Harel, David and Rosner, Roni . 1992. *Algorithmics: The Spirit of Computing*, 2nd ed. Addison-Wesley. [a readable introduction to algorithms and computing; broader scope]
- Révész, György E. 1991. *Introduction to Formal Languages*. Dover. [**L**]
- Rogers, Hartley , Jr. 1987. *Theory of Recursive Functions and Effective Computability*. MIT Press. [classic; **L**]
- Salomaa, Arto. 1985. *Computation and Automata*. Cambridge Univ. Press. [**L**]
- Traub, J. F. and Werschulz, A. G. 1998. *Complexity and Information*. Cambridge Univ. Press. [**L**]

## Logic and applied logic

- Ebbinghaus, Heinz-Dieter, Flum, Jèorg, and Thomas, Wolfgang. 1984. *Mathematical logic*. Springer-Verlag.
- Enderton, Herbert B. 2001. *A mathematical introduction to logic*, 2nd ed. Harcourt. [classic introduction; **L** (only the 1st edition, 1972)]
- Hintikka, Jaakko. 1996. *The principles of mathematics revisited*. Cambridge: Cambridge University Press. [a fresh, very critical role of FOL; discusses an alternative (still first-order) logic based on game-theoretic interpretation]
- Huth, Michael R. A. and Ryan, Mark D. 2000. *Logic in Computer Science: Modeling and Reasoning about Systems*. Cambridge Univ. Press. [intro to (semantic) model checking and (syntactic) program verification; **L**]
- Loeckx, Jacques, Ehrich, Hans-Dieter, and Wolf, Markus. 1996. *Specification of abstract data types*. Wiley. [emphasis on the logic-structure connection applied to software specification, lightly touched on in connection to the discussion of theories (Unit A2)]

## Beyond Turing computability

- Copeland, B. Jack. 2003. Hypercomputation. *Minds and Machines* 12(4):461-502. [very broad survey; an intro to the series of articles on hypercomputation]
- Copeland, B. Jack. 2000. Narrow Versus Wide Mechanisms: Including a Re-examination of Turing's Views on the Mind-Machine Use. *Journal of Philosophy* 97(1):5-32. [**L**]
- Copeland, B. Jack. 2000. Alan Turing's Forgotten Ideas in Computer Science. *Scientific American* 280(4):77-81. [**L**]
- Goldin, Dina. 2000. Persistent Turing Machines as a Model of Interactive Computation. In *Foundations of information and knowledge systems: First International Symposium, FoIKS 2000, Burg, Germany, February, 2000 (Lecture notes in computer science, 1762)*, eds. Klaus-Dieter Schewe and B. Thalheim, 116-135. Springer. [the idea of Interaction Machines (Unit D6) is taken from this article]
- Goldin, Dina and Keil, David. 2001. Interaction, Evolution, and Intelligence. In *Proceedings of the Congress on Evolutionary Computation*, Korea, May 2001.
- Israel, Navot and Goldenfeld, Nigel. 2004. Computational Irreducibility and the Predictability of Complex Physical Systems. *Physical Review Letters* 92(7):074105. [analysis of Wolfram's ideas]
- MacLennan, B. J. 2003. Transcending Turing Computability. *Minds and Machines* 13(1):3-22. [used for Ex D4]
- Milner, Robin. 1993. Elements of Interaction (Turing Award Lecture). *Communications of the ACM* 36(1):78-89. [the idea of $\pi$-calculus (Unit D6) is taken from this article; **L**]
- Shagrir, Oron and Pitowsky, Itamar. 2003. Physical Hypercomputation and the Church-Turing Thesis. *Minds and Machines* 13(1):87-101.
- Siegelmann, Hava T. 2003. Neural and Super-Turing Computing. *Minds and Machines* 13(1):103-114.
- Siegelmann, HT. 1995. Computation Beyond the Turing Limit. *Science* 268:545-548. [**L**]
- Stannet, Mike. 2003. Computation and Hypercomputation. *Minds and Machines* 13(1):115-153.
- Steinhart, Eric. 2003. Supermachines and Superminds. *Minds and Machines* 13(1):155-186.
- Thomas, Wolfgang. 1990. Automata on Infinite Objects. In *Handbook of theoretical computer science*, ed. J. van Leeuwen, 133-191. Elsevier.
- van Leeuwen, Jan and Wiedermann, Jirí. 2001. The Turing machine paradigm in contemporary computing. In *Mathematics Unlimited - 2001 and Beyond*, eds. B. Enquist and W. Schmid, 1139-1155. Springer. [a draft version of (i.e., not necessarily identical to) the listed paper available at van Leeuwen's web site: http://www.cs.uu.nl/people/jan/; used for Ex D5]
- von Neumann, John. 1967. The General and Logical Theory of Automata. In *Cerebral Mechanisms in Behavior*, ed. Lloyd A. Jeffress, 1-31. Hafner Publishing.
- Wegner, Peter. 1997. Why Interaction is More Powerful Than Algorithms. *Communications of the ACM* 40(5):80-91. [**L**]
- Wegner, Peter and Goldin, Dina. 2003. Computation Beyond Turing Machines: Seeking appropriate methods to model computing and human thought. *Communications of the ACM* 46(4):100-102. [**L**]

- Wolfram, Stephen. 2002. *A new kind of science*. Wolfram Media. [extensive work on cellular automata; **L**]


## Not classified


- Booth, Wayne C., Colomb, Gregory G., and Williams, Joseph M. 2003. *The craft of research*, 2nd ed. University of Chicago press. [excellent book on writing a research paper; basis for the connection between practical and research problems as well as the reference to "cost"/"significance" along with a problem (Unit A1); **L**]
- Gottman, John M., Murray, James D., Swanson, Catherine C., Tyson, Rebecca, and Swanson, Kristin. 2002. *The mathematics of marriage: dynamic nonlinear models*. MIT Press. [**L**]
- Kline, A. David. 1998. Introduction (to Part 4 Theory and Observation). In *Introductory readings in the philosophy of science*, eds. E. D. Klemke, Robert Hollinger, and David Wèyss Rudge, 309-315. Prometheus Books. [some useful ideas about "theories"]
- Michalewicz, Zbigniew and Fogel David B. 2000. *How to Solve It: Modern Heuristics*. Springer-Verlag. [discusses a variety of probably intractable (and other problems) from a practical point of view; **L**]
- Simon, Herbert Alexander. 1996. *The sciences of the artificial*, 3rd ed. MIT Press. [**L** (1st ed.)]
- Stewart, David and Mickunas, Algis. 1990. *Exploring phenomenology: a guide to the field and its literature*, 2nd ed. Ohio University Press.
- Suppes, Patrick. 2002. *Representation and invariance of scientific structures.* CSLI Publications. [esp. on scientific theory and the logic-structure connection]


// End

**CMSC485 (Section 2, Spring 2003) *Theory of Computation* Sample Problems**

This document contains sample problems for this semester. The problems are supposed to be ‘practical’, at least to some extent. You may find some contexts where your action might be different depending on whether or not you know the answer (and why). If you are not interested in any of these problems or know how to solve these problems, there is no point for you to take this course. On the other hand, if you find some of these problems interesting and/or relevant, this course will introduce you to the Theory of Computation so that you will be able to answer not only these questions but also related ones, which might pop up in the future. The sample problems are collected from various sources. Some are adapted or re-created for this course.

# 1. Infinite Loop Detection

Infinite loop is one of the most common bugs in your (or anyone else’) program. It would be enormously helpful if someone writes a program to detect infinite loops in a given program (as a text file). Would it be possible to write such a program?

# 2. Mayan Script

You found an ancient Mayan script that is supposed to indicate the location of hidden treasures. The only language resources in that language you have access to are: (1) an extensive list of synonyms and (2) a collection of sentences whose meaning are already known.

For example, let’s imagine that you need to translate the following sentence (obviously, not real Mayan): “koregawakarukane” with the following resources:

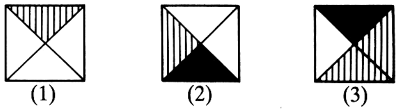
Synonym list: “korega” = “maa”, “ruka” = “nnee”, “neene” = “darona”, etc.
Sentences whose meaning are known: “maawakanneedarona”, “mosikasitarawakarukamone”, etc.

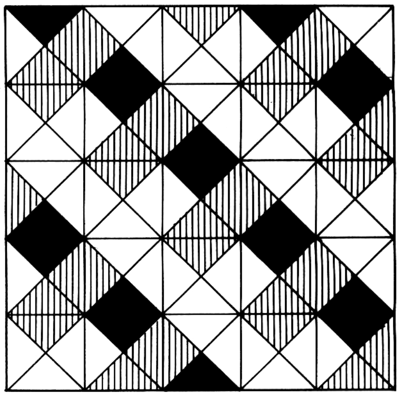
A sample session of translation by substituting synonyms would be (underlined words are replaced with *italic* words):
“koregawakarukane” → “*maa*wakarukane” → “maawaka*nnee*ne” → “maawakannee*darona*”

Then, we can tell that “koregawakarukane” means the same thing as “maawakanneedarona”, which we can understand. If there are many sentences to analyze and the synonym list is extensive, we will naturally think of doing the task with a computer. Would it be possible to write a program to solve a problem of this kind?

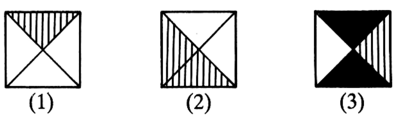
# 3. Floor Tiling

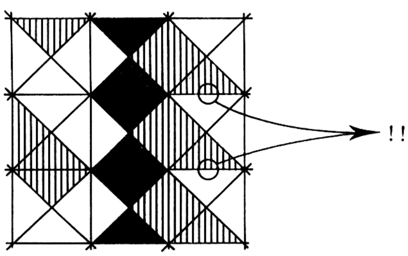
Suppose that you are searching for a house to live in and have found a one you like, except that none of the floors are covered. So, you decided to buy the house any way and finish the entire floor with a variety of tiles available at the Home Depot. You will be using *n* types of *square* tiles with distinct patterns. To demonstrate your esthetic sense, you will match the edges of the tiles as shown below (in this case, using three types of tiles). [Images from *Algorithmics* by David Harel]



Note that if you choose a wrong set of tiles, you may never be able to demonstrate your esthetic capability, as shown below.




Would it be possible to write a program to tell whether a given set of *n* types of square tiles can fill your floors with the above-mentioned condition?

## 4. Linear Programming

Consider the following problems:

- Find the cheapest combination of foods that will satisfy all your nutritional requirements
- Minimize the risk in your investment portfolio subject to achieving a certain return
- In order to mass produce complex electronic circuit boards, an almost uncountable number of holes need to be drilled. How do we route the drill (attached to a robot arm) to visit all these holes so it takes the shortest possible route and time?

All of these problem can be tackled by a technique called ‘linear programming’. Roughly, linear programming is an approach to set up *k* linear inequalities with *n* variables and find an assignment for the variables that satisfies all the inequalities.

As many problems are rather complex, researchers have been solving them mainly through case-by-case approaches.  Would there be a general way to solve all of these problems?

## 5. Ambiguity Detection

A simple rewriting system can be used to characterize a certain language (set of strings).  For example, consider the following set of rewrite rules (can be used to represent programming language syntax):

$A \to x\,A\,y$ (Rule 1)
$A \to x\,y$ (Rule 2)
$A \to x\,x\,y\,y$ (Rule 3)

Starting from the symbol *A*, we can generate all sorts of strings of the form *x*...*xy*...*y* where the number of *x*'s and that of *y*'s are the same.  For example, applying Rule 1 twice and Rule 2 twice would result in an output '*xxxyyy*' as shown below:

$$A \to x\,A\,y \to x\,x\,A\,y\,y \to x\,x\,x\,y\,y\,y$$

One tricky point about this type of formulation is that the system can be 'ambiguous'.  That is, there may be more than one way of generating the same result.  For example, the above system can generate '*xxxyyy*' also by applying Rule 1 once and then Rule 3 once.  Naturally, ambiguous systems could cause a lot of trouble for precise representation of a language and also for processing the language.  Would there be a general way to detect whether this type of system is ambiguous?

## 6. Program Verification

One of the software industry's biggest concern is how to check whether their products are really correct with respect to the specification that the developer and the user both agreed.  According to a variety of sources, inability to do so will cost us an incredible amount of money in the future (well, this must be already happening).  As you know, *generality* is a prime concern in Computer Science.  So, why don't we write a single program that could verify whether the program correctly solves a problem with respect to a given specification?  But would it be possible?

## 7. Arithmetic System

One of the great achievements in logic is that it can formalize a wide variety of systems.  For example, the mathematical system of arithmetic can be formalized in first-order logic in a consistent (roughly, meaningful) way.  Its basic components include formulas such as "for any $x$, $x + 0 = x$" and "for any $x$, $x \times 0 = 0$".  In this system, we can write all sorts of formulas that are representative of arithmetic.  One property that a good logic system must have is that all 'true' formulas can be proven as a series simple steps.  But here is a big question.  Would it be really possible to prove all true formulas in the arithmetic system?

## 8. Hilbert's Tenth Problem

A great mathematician Hilbert once asked: whether or not a given polynomial equation with integer coefficients has a solution in integers. Is this in general solvable?

## 9. Program Elegance

Many programmers may be trying to write 'elegant' programs (only if they have time, which is of course never available). But could we ever tell whether a particular program is elegant?

## 10. Functional Programming Languages

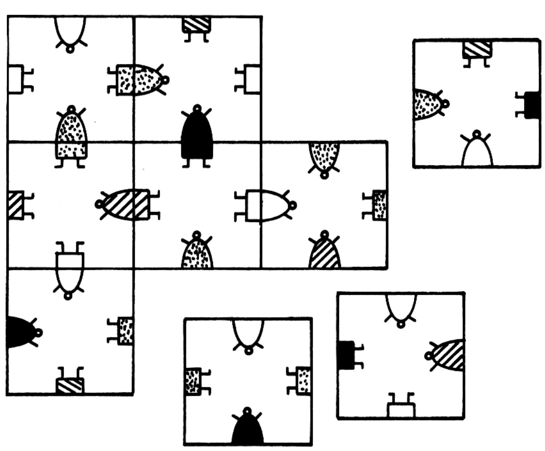
LISP/Scheme, ML, and Haskell are known as 'functional' programming languages. They differ from more popularized 'procedural' programming languages such as Basic and C. We can often observe that programs in a functional language look rather different from those in procedural languages, even for doing the same task. For example, you might never see a loop, only recursions. However, both of these language groups seem to be able to do pretty much the same thing. Could we justify that they can actually solve exactly the same class of computational problems?

## 11. Cryptography

Public key cryptography employs two types of keys, public and private. The basic idea behind the use of two keys is that (A) the public key can easily be obtained from the private key and (B) the private key can***not*** easily be obtained from the public key. For example, RSA cryptosystem satisfies these conditions. However, cryptology is a never ending battle. We will need to have many other cryptosystems for coming years (in fact, also right now). Then, how can we draw the line between 'easy' and 'not-easy' computation in a systematic manner?

## 12. Monkey Puzzle

The following is a snapshot of a game to place 9 square cards so that pictures will match (note that the cards cannot be rotated). [Image from *Algorithmics* by David Harel]



In general, we can think of a puzzle in any $n \times n$ size. How fast could we solve such a puzzle?

## 13. Digital Circuits

To design digital logic circuits, we need to analyze the relation between inputs and the output. For example, the following formula represents a circuit with four inputs and one output, each of which could take either 0 or 1 ('+' for OR, '·' for AND, and ''' for NOT).

output = $(i_1' + i_2 + i_3 + i_4) \cdot (i_1' + i_2 + i_3 + i_4') \cdot (i_1' + i_2' + i_3 + i_4')$

Given an arbitrary formula, we can surely find out whether some combination of inputs would result in output 1, by checking all the combinations (you should be able to tell how long it would take to compute all the results for the *n*-variable case). However, we would surely hope that there is a better way. How fast could we compute the output in general?

## 14. Professor Assignment

It is not a simple task of assigning all the CS faculty members to all the courses offered by the CS department. Well, it is actually not that complicated at TCNJ. But what about a large English department in a giant state university? Imagine that as a part of your work study, you are asked to write a program to do the scheduling for a large department. How fast in general could your program schedule all the faculty members?

## 15. Knapsack Problem

There will be an excursion tomorrow. You will need to pack all of your junk food packages in your knapsack. However, you realized that not all of them would fit in the knapsack. So, you want to fit as many as possible while the total cost of your junk foods is maximized (to impress your pals). How fast could you find out the solution given an arbitrary scenario?

## 16. Cross-Country Interviews

Suppose that you are invited for interviews (graduate schools or industry jobs) in a number of cities in the US. Unfortunately, they do not pay for your travel. So, you will need to minimize the expense. How fast in general could you find the best way to travel all the cities?

## 17. CPU Register Allocation

CPU is no doubt the 'brain' of a computer. We want to cram as many things as possible in CPU's. However, we also need to consider the cost of doing so. In practice, we have to settle down at some cost-performance level. The cost-performance trade off also applies to the number of registers in a CPU. In order to find the optimal number of registers for a future CPU, we want to write a program to find the minimal number of registers for a collection of popular programs. What would be the performance of such a program?

## 18. Map Coloring

It has been shown that with four distinct colors, we can color any map so that the neighboring countries (or whatever political boundaries) do not share the same color. With three colors, we may or may not be able to do the same thing. How fast in general could we find out whether a map can be 3-colored?

## 19. Time-Space Tradeoffs

If our computers had an infinite amount of memory, we could load all the programs and run them without accessing the hard disk. This would eliminate our frustration with loading time. However, this will remain as dream. Then, we need to compromise the use of space (memory, etc.) and the processing time. Can we say anything general about the time-space tradeoff?

## 20. *Respectively* in English

In English, you can say something like: *a*, *b*, *c*, and *d* are the lower case of *A*, *B*, *C*, and *D*, respectively. In general, there is no limit to the number of items that are connected in this manner. What would be the simplest mechanism that can process the correspondence of multiple sequences like this?

## 21. Programming Language Parsing #1

Modern programming languages allow an arbitrary level of nesting. You do not need to write a LISP program to be baffled by an insane number of '(' and ')' as well as all sorts of other nesting constructions. What would be the simplest representation (i.e., that can*not* represent anything more complicated) for this type of conditions? What would be the simplest mechanism that can handle the 'nesting' property of modern programming languages? In addition, are there properties of programming languages that would require more than handling nesting?

## 22. Programming Language Parsing #2

Suppose that we know the answer to the previous problem. In other words, we know how to characterize the set of all valid programs in terms of program structure. Then, given an arbitrary program, can we always identify whether it belongs to the set? In other words, is parsing possible? Of course all sorts of programming languages are being parsed by all sorts of compilers and interpreters. The question here is more general because we are talking about all programming languages with, e.g., the nesting property, including those we have never seen.

## 23. Password Screening

When you obtained an account for a host computer, you might have been asked to choose a password that satisfies the conditions: (i) at least six characters, (ii) must include at least one symbol or numeral, (iii) must include at least one upper case character, etc. It shouldn't be difficult to check all these conditions. But in order to maintain the security of the system, this kind of screening must definitely be done by a program. What would be the simplest representation (i.e., that can*not* represent anything more complicated) for this type of conditions? What would be the simplest mechanism that can handle this and similar conditions in a systematic manner?

<End>

# CSC460 (Spring 2005) Module A Evaluation Form

| Name | Self-evaluation (A, B, C possibly with +/–) | |
|---|---|---|
| | Adjustment by the instructor | |

## Evaluation Materials (Portfolio)

Your evaluation materials (referred to as "**portfolio**," and to be placed in the provided manila folder) consist of the following **Items**:

1. This form (must be filled out; see the instructions below)
2. Word-processed **supporting notes** responding to the instructions in this form (except for the materials completed during the evaluation workshop)
3. Take-home **exercises** (including the comprehensive exercise), chronologically ordered
4. Materials completed during the **evaluation workshop** (to be explained in class)

## Self-Evaluation Procedure

During the module, before the evaluation workshop

- You regularly examine the learning goals check list (included below).
- If you think you satisfy a criterion, (i) place a check mark in the box (☑) at the end of the criterion, and (ii) in your supporting notes, explain *how* you satisfied the criterion in a way the instructor can be convinced. That is, you are expected to explain the *process*, not just the consequence.
- If you believe that you already wrote your response to a certain criterion in an earlier exercise, you can simply refer to that exercise (which must be included in your portfolio).

During the evaluation workshop (preliminary)

- At the beginning of the evaluation workshop, you must have **Items 1 through 3** *as hard copies*, as well as blank sheets for **Item 4**.
- There will be an in-class, open-ended module review exercise to check your understanding of the learning goals. Although your answers will not be graded per se, they may be used to validate your supporting notes. For example, if you thought you achieved the learning goals and could still not be able to respond to the review exercise well, you will need to improve your self-evaluation (Performance Goal 7).
- You will have an opportunity to share your portfolio with other students.
- Finally, you will write a reflective essay revealing your thought process during the evaluation workshop, and assign yourself a grade based on the course grading scheme (included at the end of this sheet).
- At the end of the session, you will submit your portfolio.

After the evaluation workshop

- If your written justification for a criterion is convincing, the instructor will also place a check mark next to yours.
- If necessary, the instructor will adjust your grade.
- Normally, your portfolio will be returned in the next class meeting.

## Learning Goals Checklist (the goals not pursued in this module are "grayed")

In your **supporting notes**, clearly identify the criteria, e.g., **C1*a*** (for Content Goal 1 Criterion *a*), **P5*b*** (for Performance Goal 5 Criterion *b*), referring to the labels below.

### *Content Goals*

1. Practical problems can often be transformed into research and computational problems. Every problem is associated with cost/significance, which is relative to the evaluator. Computational problems can be represented as a set, readily available as the input to computational processing. [**problem**]
   *a.* Understood the connections and differences among practical, research, and computational problems, as well as the notion of cost/significance. ☐
   *b.* Explained why a research problem can be transformed into a set. ☐
   *c.* [optional; if necessary] Reviewed concepts in discrete math (sets, relations, functions, mathematical structures, logic; ref. http://www.tcnj.edu/~komagata/cmsc210/03f/Topics.pdf). (☐)
2. A theory is a potentially infinite, consistent body of knowledge which can be systematically derived from a small number of abstract principles. The gap between the principles and the entire information is the source of the theory's predictive power. Also being abstract, a theory can be applied to a broad range of phenomena, which might appear distinct. [**theory**]

   *a*. Understood the notion of theory, referring to its “predictability” and “applicability,” using *your own* examples. ☐
   *b*. Explained how Theory of Computation could impact your career involving computation. ☐
3. Interactive computation subsumes algorithmic computation, but not vice versa. That is, there is a qualitative difference between these two modes of computation. [**interactive computation**] [not in this module]
4. The algorithmic notion of computation can be represented in a variety of equivalent forms, which define a bounded class of sets. That is, there is a limit to algorithmic computation. [**computability**]
   *a*. Understood the basic mechanism of Turing machine, partly through the use of a simulator. ☐
   *b*. Explained the connection between computational problems represented as sets and the use of Turing machines to process them. ☐
   *c*. Explained following the notions using a schematic diagram: decidability, undecidability, TM-recognizability, unsolvability, semi-decidability. ☐
5. The computable class of sets contain a hierarchy of proper subsets which can be characterized by distinct grammars and automata. [**formal languages and automata**] [not in this module]
6. The practicality of an algorithm depends on its complexity relative to the input data size. [**complexity**] [not in this module]
7. Power set, also encompassing the distinction between determinism and nondeterminism, can introduce discontinuity with respect to computability and complexity. [**power set**] [not in this module]

***Performance Goals***

1. Identify real-world problems which are relevant to the student’s life and can be tackled by computational means. [**awareness**] [combined with other goals]
2. Transform real-world problems into research problems, and then into computational problems, along with the analysis of the cost/significance of a problem. [**transformation**]
   *a*. Understood the process through exercises. [simply refer to successfully-completed exercises] ☐
3. Analyze computational problems with respect to interactivity, computability, language/automata hierarchy, and complexity hierarchy. Then, evaluate the analysis with respect to its usefulness, correctness, and accuracy. [**analysis/evaluation**]
   *a*. Analyzed the computability (e.g., decidability) of a variety of problems including your own, using Turing machine as a model of computation. ☐
4. Respect, analyze, and give constructive criticisms to the ideas in the literature and those expressed by other people. [**critical attitude**] [not in this module]
5. Express ideas orally and in writing, in a manner clearly understood by other students (with equivalent background). Explain your own ideas orally and in writing, clearly and logically. Revise the ideas, reflecting the feedback from other students and the instructor. [**communication**]
   *a*. Completed all the exercises (take-home and in-class). ☐
   *b*. Responded to the other students’ and the instructor’s comments (e.g., on your exercises). ☐
6. Take initiative in both independent and group activities. Also extend the domain of theoretical inquiry beyond the scope prepared by the instructor. [**initiative**]
   *a*. Regularly contributed to class *and* group discussions/activities. ☐
   *b*. Regularly examined the evaluation criteria and placed check marks on this evaluation form. ☐
7. Reflect upon the student’s own thinking process and assess the student’s own performance relative to the content and performance goals. [**reflection**]
   *a*. Was able to reflect upon your experience in this module through the activities during the evaluation workshop. [during the eval workshop]
   *b*. Self-evaluated your achievements *accurately*. [during the eval workshop]

## Self-Evaluation Criteria

At the end of the module evaluation workshop, propose your grade based on the following scheme (possible qualification with +/–):

- Grade A: Achieved all the learning goals relevant to the module
- Grade B: Achieved almost all the learning goals (except for one or two evaluation criteria) relevant to the module
- Grade C/Pass: Achieved most of the learning goals relevant to the module

// End

# CSC460 (Spring 2005) Module B Evaluation Form

| Name | Self-evaluation (A, B, C possibly with +/–) | |
|---|---|---|
| | Adjustment by the instructor | |

## Evaluation Materials (Portfolio)

Your evaluation materials (referred to as "**portfolio**," and to be placed in the provided manila folder) consist of the following **Items**:

1. This form (must be filled out; see the instructions below)
2. Word-processed **supporting notes** responding to the instructions in this form (except for the materials completed during the evaluation workshop)
3. Take-home **exercises** (including the comprehensive exercise), chronologically ordered
4. Materials completed during the **evaluation workshop** (to be explained in class)

## Learning Goals Checklist (the goals not pursued in this module are "grayed")

In your **supporting notes**, clearly identify the criteria, e.g., **C1*a*** (for Content Goal 1 Criterion *a*), **P5*b*** (for Performance Goal 5 Criterion *b*), referring to the labels below.

### *Content Goals*

1. Practical problems can often be transformed into research and computational problems. Every problem is associated with cost/significance, which is relative to the evaluator. Computational problems can be represented as a set, readily available as the input to computational processing. [**problem**] ...... [not in this module]
2. A theory is a potentially infinite, consistent body of knowledge which can be systematically derived from a small number of abstract principles. The gap between the principles and the entire information is the source of the theory's predictive power. Also being abstract, a theory can be applied to a broad range of phenomena, which might appear distinct. [**theory**] ...... [not in this module]
3. Interactive computation subsumes algorithmic computation, but not vice versa. That is, there is a qualitative difference between these two modes of computation. [**interactive computation**]
   *a.* Was able to point out the limitations of the traditional, algorithmic approach to "computability." ...... ☐
4. The algorithmic notion of computation can be represented in a variety of equivalent forms, which define a bounded class of sets. That is, there is a limit to algorithmic computation. [**computability**]
   *a.* Understood how to create a universal TM, including how to represent a TM as the input on the tape. ...... ☐
   *b.* Understood when *and* how to use the diagonalization technique. ...... ☐
   *c.* Can explain logically that the halting problem is semi-decidable and infinite-loop detection is ~~unsolvable~~ non-TM-recognizable. ...... ☐
   *d.* Understood how to use "reduction" to (i) establish equivalence among models and (ii) analyze properties applied to different problems. ...... ☐
   *e.* Understood the description and the significance of Church-Turing thesis. ...... ☐
   *f.* Can explain (i.e., teach) this goal to CS students outside this class. ...... ☐
5. The computable class of sets contain a hierarchy of proper subsets which can be characterized by distinct grammars and automata. [**formal languages and automata**] ...... [not in this module]
6. The practicality of an algorithm depends on its complexity relative to the input data size. [**complexity**] ...... [not in this module]
7. Power set, also encompassing the distinction between determinism and nondeterminism, can introduce discontinuity with respect to computability and complexity. [**power set**]
   *a.* Understood (i) that the power set is always "larger" than the original set and (ii) why the power set of a countable set is uncountable. ...... ☐

### *Performance Goals*

1. Identify real-world problems which are relevant to the student's life and can be tackled by computational means. [**awareness**] ...... [combined with other goals]
2. Transform real-world problems into research problems, and then into computational problems, along with the analysis of the cost/significance of a problem. [**transformation**] ...... [not in this module]

3. Analyze computational problems with respect to interactivity, computability, language/automata hierarchy, and complexity hierarchy. Then, evaluate the analysis with respect to its usefulness, correctness, and accuracy. [**analysis/evaluation**] ........................................ [combined with content goals]
4. Respect, analyze, and give constructive criticisms to the ideas in the literature and those expressed by other people. [**critical attitude**]
   - *a.* *Critically* analyzed the following points: (i) the use of the infinite tape in TM [You may assume that every concrete object is finite.], (ii) the diagonalization technique, (iii) no tolerance to errors as the standard position in the Theory of Computation, (iv) the lack of interaction in TM, (v) any other aspects ........ ☐
   - *b.* *Critically* analyzed the usefulness of the "computability" area of the (traditional) Theory of Computation ☐
   - *c.* *Critically* analyzed the course materials/organization esp. in connection to how you could learn most effectively ........................................ ☐
5. Express ideas orally and in writing, in a manner clearly understood by other students (with equivalent background). Explain your own ideas orally and in writing, clearly and logically. Revise the ideas, reflecting the feedback from other students and the instructor. [**communication**]
   - *a.* Completed all the exercises on time (take-home and in-class). ........................................ ☐
   - *b.* Responded to the other students' and the instructor's comments (e.g., on your exercises). ........ ☐
6. Take initiative in both independent and group activities. Also extend the domain of theoretical inquiry beyond the scope prepared by the instructor. [**initiative**]
   - *a.* Chose a research question for the mini research project and started to explore it. ........................ ☐
   - *b.* Noted any other aspects relevant to this goal. ........................................ ☐
7. Reflect upon the student's own thinking process and assess the student's own performance relative to the content and performance goals. [**reflection**]
   - *a.* Was able to reflect upon your experience in this module through the activities during the evaluation workshop. ........................................ [during the eval workshop]
   - *b.* Self-evaluated your achievements *accurately*. ........................................ [during the eval workshop]

## Self-Evaluation Criteria

At the end of the module evaluation workshop, propose your grade based on the following scheme (possible qualification with +/–):

- Grade A: Achieved all the learning goals relevant to the module
- Grade B: Achieved almost all the learning goals (except for one or two evaluation criteria) relevant to the module
- Grade C/Pass: Achieved most of the learning goals relevant to the module

// End

# CSC460 (Spring 2005) Module C Evaluation Form

| Name | Self-evaluation (A, B, C possibly with +/–) | |
|---|---|---|
| | Adjustment by the instructor | |

## Evaluation Materials (Portfolio)

Your evaluation materials (referred to as "**portfolio**," and to be placed in the provided manila folder) consist of the following **Items**:

1. This form (must be filled out; see the instructions below)
2. Word-processed **supporting notes** responding to the instructions in this form (except for the materials completed during the evaluation workshop)
3. Take-home **exercises** (including the comprehensive exercise), chronologically ordered
4. Materials completed during the **evaluation workshop** (to be explained in class)

## Learning Goals Checklist (the goals not pursued in this module are "grayed")

In your **supporting notes**, clearly identify the criteria, e.g., **C1*a*** (for Content Goal 1 Criterion *a*), **P5*b*** (for Performance Goal 5 Criterion *b*), referring to the labels below.

***Content Goals***

1. Practical problems can often be transformed into research and computational problems. Every problem is associated with cost/significance, which is relative to the evaluator. Computational problems can be represented as a set, readily available as the input to computational processing. [**problem**] [not in this module]
2. A theory is a potentially infinite, consistent body of knowledge which can be systematically derived from a small number of abstract principles. The gap between the principles and the entire information is the source of the theory's predictive power. Also being abstract, a theory can be applied to a broad range of phenomena, which might appear distinct. [**theory**] [not in this module]
3. Interactive computation subsumes algorithmic computation, but not vice versa. That is, there is a qualitative difference between these two modes of computation. [**interactive computation**] [not in this module]
4. The algorithmic notion of computation can be represented in a variety of equivalent forms, which define a bounded class of sets. That is, there is a limit to algorithmic computation. [**computability**] [not in this module]
5. The computable class of sets contain a hierarchy of proper subsets which can be characterized by distinct grammars and automata. [**formal languages and automata**]
   - *a.* Understood that there is a hierarchy of TM-recognizable languages that can be specified and processed by the corresponding grammars and automata, respectively [Chomsky hierarchy]. ☐
   - *b.* Understood that TM-recognizable languages can be specified by unrestricted grammars (rewriting system), which are equivalent to TMs. Also understood why this class is not useful in practice. ☐
   - *c.* Understood that context-free languages (CFLs) can be specified by context-free grammars (CFGs) and processed by push-down automata (PDAs). Also understood when/how to use this class to analyze problems. ☐
   - *d.* Understood that the deterministic subset of CFLs is an important class that supports the backbone of programming languages, which can be processed by deterministic PDAs (DPDAs). ☐
   - *e.* Understood that regular languages (regular sets) can be specified by regular expressions (RegExps) and processed by finite-state automata (FSAs). Also understood when/how to use this class to analyze problems. ☐
   - *f.* Understood how to use the pumping lemma to show that a language is *not* regular. ☐
   - *g.* Understood how to use the pumping lemma for CFLs to show that a language is *not* context-free. Also understood that certain properties of CFLs are undecidable. ☐
   - *h.* Could explain (i.e., teach) this goal to CS students outside this class. ☐
6. The practicality of an algorithm depends on its complexity relative to the input data size. [**complexity**] [not in this module]
7. Power set, also encompassing the distinction between determinism and nondeterminism, can introduce discontinuity with respect to computability and complexity. [**power set**]
   - *a.* Understood that nondeterminism can be represented as the power set of the possible states. ☐
   - *b.* Understood why nondeterminism affects the power of PDAs but neither TMs nor FSAs. ☐

***Performance Goals***

1. Identify real-world problems which are relevant to the student's life and can be tackled by computational means. [**awareness**] [combined with other goals]
2. Transform real-world problems into research problems, and then into computational problems, along with the analysis of the cost/significance of a problem. [**transformation**] [not in this module]
3. Analyze computational problems with respect to interactivity, computability, language/automata hierarchy, and complexity hierarchy. Then, evaluate the analysis with respect to its usefulness, correctness, and accuracy. [**analysis/evaluation**] [combined with content goals]
4. Respect, analyze, and give constructive criticisms to the ideas in the literature and those expressed by other people. [**critical attitude**]
   *a.* *Critically* analyzed the usefulness of the "languages/automata" area of the (traditional) Theory of Computation, esp. in connection to your ability to choose the minimal specification/process for the given problem. ☐
5. Express ideas orally and in writing, in a manner clearly understood by other students (with equivalent background). Explain your own ideas orally and in writing, clearly and logically. Revise the ideas, reflecting the feedback from other students and the instructor. [**communication**]
   *a.* Completed all the exercises on time (take-home and in-class). ☐
   *b.* Made conscious efforts to promote transfer of learning among students, esp. during in-class exercises (e.g., explained what you understood to other students and learned what you didn't understand from other students). ☐
6. Take initiative in both independent and group activities. Also extend the domain of theoretical inquiry beyond the scope prepared by the instructor. [**initiative**]
   *a.* Continued to analyze your own problem and mini research project, by applying the notion of "modularity" and identifying the simplest mechanisms for the component modules. ☐
   *b.* Noted any other aspects relevant to this goal. ☐
7. Reflect upon the student's own thinking process and assess the student's own performance relative to the content and performance goals. [**reflection**]
   *a.* Was able to reflect upon your experience in this module through the activities during the evaluation workshop. [during the eval workshop]
   *b.* Self-evaluated your achievements *accurately*. [during the eval workshop]

## Self-Evaluation Criteria

At the end of the module evaluation workshop, propose your grade based on the following scheme (possible qualification with +/–):

- Grade A: Achieved all the learning goals relevant to the module
- Grade B: Achieved almost all the learning goals (except for one or two evaluation criteria) relevant to the module
- Grade C/Pass: Achieved most of the learning goals relevant to the module

// End

# CSC460 (Spring 2005) Module D Evaluation Form

| Name | Self-evaluation (A, B, C possibly with +/–) | |
|---|---|---|
| | Adjustment by the instructor | |

## Evaluation Materials (Portfolio)

Your evaluation materials (referred to as "**portfolio**," and to be placed in the provided manila folder) consist of the following **Items**:

1. This form (must be filled out; see the instructions below)
2. Word-processed **supporting notes** responding to the instructions in this form (except for the materials completed during the evaluation workshop)
3. Take-home **exercises** (including the comprehensive exercise), chronologically ordered
4. Materials completed during the **evaluation workshop** (to be explained in class)

## Learning Goals Checklist (the goals not pursued in this module are "grayed")

In your **supporting notes**, clearly identify the criteria, e.g., **C1*a*** (for Content Goal 1 Criterion *a*), **P5*b*** (for Performance Goal 5 Criterion *b*), referring to the labels below.

### *Content Goals*

1. Practical problems can often be transformed into research and computational problems. Every problem is associated with cost/significance, which is relative to the evaluator. Computational problems can be represented as a set, readily available as the input to computational processing. [**problem**] [not in this module]
2. A theory is a potentially infinite, consistent body of knowledge which can be systematically derived from a small number of abstract principles. The gap between the principles and the entire information is the source of the theory's predictive power. Also being abstract, a theory can be applied to a broad range of phenomena, which might appear distinct. [**theory**] [not in this module]
3. Interactive computation subsumes algorithmic computation, but not vice versa. That is, there is a qualitative difference between these two modes of computation. [**interactive computation**]
   - *a.* Understood the effects (and limitations) of parallel computation with respect to the three subareas of the traditional Theory of Computation (in a sense as a preliminary to interactivity). ☐
   - *b.* Understood what kind of problems can*not* be adequately represented by TMs. ☐
   - *c.* Understood the basics of super-Turing computation (more "powerful" than TMs) including its significance. ☐
   - *d.* Was able to speculate where the theoretical underpinning of computer science should be heading, in order to offer robust analyses of a variety of computational problems. ☐
4. The algorithmic notion of computation can be represented in a variety of equivalent forms, which define a bounded class of sets. That is, there is a limit to algorithmic computation. [**computability**] [not in this module]
5. The computable class of sets contain a hierarchy of proper subsets which can be characterized by distinct grammars and automata. [**formal languages and automata**] [not in this module]
6. The practicality of an algorithm depends on its complexity relative to the input data size. [**complexity**]
   - *a.* Reviewed how to interpret the big *O* notation [game-theoretically and using the on-line graphing tool (http://www.tcnj.edu/~komagata/Graphing)]. ☐
   - *b.* Understood that exponential growth with respect to the input data size (time complexity) is considered impractical ("intractable"), through examples. ☐
   - *c.* Understood the class of "nondeterministic polynomial" (NP) problems, through examples. Also understood (i) why there are so many NP problems and (ii) why NP problems are essential for computer security. ☐
   - *d.* Understood the notion of "polynomial time reducibility" and its impact on relating problems. ☐
   - *e.* Understood the class of "nondeterministic polynomial complete" (NPC) problems, through examples. Also understood (i) the basics of how to show that a problem is in NPC and (ii) why this class is important. ☐
   - *f.* Understood the essential difference between time and space complexity, as well as the hierarchy involving various time/space complexity classes. ☐
   - *g.* Could explain (i.e., teach) this goal to CS students outside this class. ☐

7. Power set, also encompassing the distinction between determinism and nondeterminism, can introduce discontinuity with respect to computability and complexity. [**power set**]
   *a.* Understood that power set introduces exponential growth, which could lead to intractability. ☐
   *b.* Gained insight into the meaning of power set (in contrast to just knowing the formal definition), which is reflected in all three subareas of the traditional Theory of Computation (and also in reality). ☐

***Performance Goals***

1. Identify real-world problems which are relevant to the student's life and can be tackled by computational means. [**awareness**] [combined with other goals]
2. Transform real-world problems into research problems, and then into computational problems, along with the analysis of the cost/significance of a problem. [**transformation**] [not in this module]
3. Analyze computational problems with respect to interactivity, computability, language/automata hierarchy, and complexity hierarchy. Then, evaluate the analysis with respect to its usefulness, correctness, and accuracy. [**analysis/evaluation**] [combined with content goals]
4. Respect, analyze, and give constructive criticisms to the ideas in the literature and those expressed by other people. [**critical attitude**]
   *a.* *Critically* analyzed the usefulness of the "complexity" area of the (traditional) Theory of Computation, i.e., Content Goal 6. ☐
   *b.* *Critically* analyzed other students' practicum presentation, esp. in connection to aspects relevant to the Theory of Computation. [**to be assessed by the instructor on 4/27/05; no need to write**]
5. Express ideas orally and in writing, in a manner clearly understood by other students (with equivalent background). Explain your own ideas orally and in writing, clearly and logically. Revise the ideas, reflecting the feedback from other students and the instructor. [**communication**]
   *a.* Completed all the exercises on time (take-home and in-class). In particular, clearly articulated the understanding of the Theory of Computation in the mini research paper [Comprehensive Exercise]. ☐
6. Take initiative in both independent and group activities. Also extend the domain of theoretical inquiry beyond the scope prepared by the instructor. [**initiative**]
   *a.* Continued to analyze your own problem and mini research project, by applying the notion of "complexity," i.e., Content Goal 6. ☐
   *b.* A symbol (form) can mean different things (content) to different people, depending on the context [ref. Unit A2]. An analogous idea applies to a course as well. For example, this course could mean different things to different people; a positive attitude might lead to a positive outcome. What kind of initiative did you take to make your experience in this course positive? ☐
7. Reflect upon the student's own thinking process and assess the student's own performance relative to the content and performance goals. [**reflection**]
   *a.* Was able to reflect upon your experience in this module through the activities during the evaluation workshop. [during the eval workshop]
   *b.* Self-evaluated your achievements *accurately*. [during the eval workshop]

## Self-Evaluation Criteria

At the end of the module evaluation workshop, propose your grade based on the following scheme (possible qualification with +/–):

- Grade A: Achieved all the learning goals relevant to the module
- Grade B: Achieved almost all the learning goals (except for one or two evaluation criteria) relevant to the module
- Grade C/Pass: Achieved most of the learning goals relevant to the module

// End

## CSC460 Theory of Computation

*Today*

- Understand the course learning goals
- Understand the course organization
  - Syllabus
  - On-line resources
- Identify the first things to do


Intro example


## This Course

- Theme: Apply the Theory of Computation
  - Application: Your computational needs
  - Theory: A small number of principles
- Approach: Problems, Transform, Compute, Discuss, and Evaluate
  - Emphasis on discussion during class
  - Take-home exercise after every class
  - No memorization; no exams


Initial survey: General, Technical


Section 1

## Course Learning Goals

- Content goals
  - Understand ... (ideas)
- Performance goals
  - Being able to do ... (actions)

might appear complicated initially...

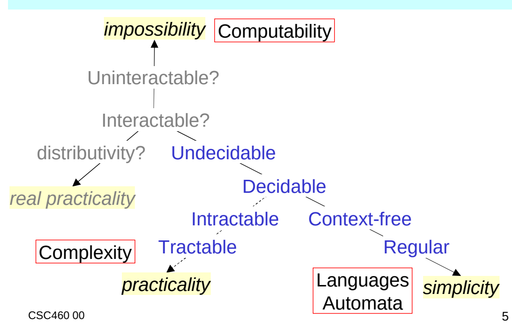


## Content Goals: Understand

1. Problem
2. Theory
3. Interactive computation
4. Computability
5. Formal languages and automata
6. Complexity
7. Power set



## Preview: Theory of Computation






## Performance Goals: Do

1. Awareness
2. Transformation
3. Analysis/evaluation
4. Critical attitude
5. Communication
6. Initiative
7. Reflection

Section 2

## Course Organization

- Syllabus
- On-line resources
  - Course web pages
  - Instructor's page for students
  - SOCS: E-mail, discussion, grades
- Textbook
- Schedule, Course modules

Alignment of goals - assessment - activities



## Course Modules

A. Problems, Theory, Computability (1)
B. Computability (2)
C. Formal Languages and Automata
D. Complexity

Each module: Roughly a quarter of the semester (6-7 class meetings)



## Assessment

- Module evaluation materials (portfolio)
  1. Evaluation form
  2. Support notes (word-processed)
  3. Take-home exercises (and in-class notes)
  4. Materials completed during Evaluation Workshop

Carry these materials to class

Work: Collaborate
Assessment: **Individual**



## Class Meetings

- Clarification of the learning goals, etc.
- Survey (occasionally)
- Presentation of examples, cases, etc.
- Discussion of ideas, principles, etc.
- Class and group discussions
- Practice using the evaluation form
- Explanation of the take-home exercises
- Evaluation workshop



## Current Plan and Flexibility

- The current plan based on my experience
  - Observed some positive aspects in other courses
- This class: A small number of presumably motivated students
  - Can modify the plan fairly flexibly
  - Your suggestions on course organization welcome



## Self-Introduction

- Name
- How you want to be called
- Why you are here (i.e., taking this course or was at least interested in this course)

# First Things to Do

*By the next class meeting on Friday*

- Exercise 00 [all exercises to be put on-line]
  - Part 1 (essay): Your own problem(s) that can be solved computationally Class discussion next time
  - Part 2: Course Preview
    - Read and *understand* the syllabus
    - Read Module A Evaluation Form
    - Visit the course page [try most links]
      - Read the instructor's *page for students*
    - Set SOCS discussion e-mail notification

Name: ______________________

# Exercise 00, 1/18/05

Note: All take-home exercises are due at the beginning of the following class meeting (unless otherwise stated). For this exercise, the due date is 1/21/05, this coming Friday. At that time, submit this sheet with other required components.

Most of the take-home exercises in this course will be open-ended. This reflects the belief of the instructor that "important" real-world problems are often open-ended (in other words, problems with well-defined answers are not so important), and that students must practice facing such problems in regular courses as well as other opportunities such as research and internship. If you feel more comfortable with being able to present the "expected" answers to well-defined problems, these exercises will be opportunities to develop a different attitude to problems. Such an attitude would actually be more useful when you begin a career after graduation, be it in the industry or graduate work.

## Part 1: Your Own Problem(s)

Probably the best way to learn and appreciate the power of Theory of Computation is to apply it to computational problems that you care. In addition, it would be more effective if we discuss issues that you are familiar with and/or interested in. Thus, it is *extremely important* that you think and write about problems you know at this stage.

**Task:** Write an essay about one or more of your own problem(s) that can be solved computationally (i.e., using a computer or defining a computational process such as algorithm). Try to be reasonably detailed so that the reader can understand your problem(s) well. Optionally, you may want to analyze certain computational aspects (e.g., whether computable within a reasonable amount of time) as much as you can. Be flexible and creative. Look around and think carefully. In addition, be prepared to discuss your problem in class.

Instructions/Notes:
1. Word-process your essay and submit a hard copy at the designated time. ........ ☐
2. Include: basic course info, exercise ID (00 for this one), your name, and date. ........ ☐
3. Do not include a title page. ........ ☐
4. Try to be concise (no extraneous information) and clear (understandable by other college students). ........ ☐
5. Include page numbers, if more than one page. ........ ☐
6. No requirements on font selection/size, line space, margin, or the number of pages. Apply your common sense.

## Part 2: Course Preview

Just to make sure that you understand the course organization and content well, do all of the following things (sorry to be too prescriptive; you will have more flexibility as we go):

**Tasks**:
1. Read the syllabus.

   *a.* If you understand it *completely*, check the box at right. ☐
   *b.* If you have questions, check the box and *list them on the other side of this sheet*. ☐
2. Read Module A Evaluation form.
   *a.* If you understand the general idea about evaluation, check the box at right. ☐
   *b.* If you have questions, check the box and *list them on the other side of this sheet*. ☐
3. Visit the course page and most of the links including the instructor's page for students. ☐
   *a.* If you understand the "Course work" section of it, check the box at right. ☐
   *b.* If you have questions, check the box and *list them on the other side of this sheet*. ☐
4. Log on to SOCS and set discussion board e-mail notification. ☐

Survey: Time spent between classes for this course (this exercise, etc.): _______________

// End

## What to do with your problems?



## Unit A1: Overview

- Discuss your own problems
- Analyze the process of going from problems to solutions
- Identify different types of problems
- Discuss how to represent problems
- Practice transforming problems
- Preview Exercise A1 "Using a Theory"

Questions about the course? (Ex00 Part 2)



## Exercise 00 Part 1

- Your own problems
- Analysis of computational aspects (optional)
  - Can obtain an answer?
  - Can do it with available resources?
  - Can do it within a reasonable time?
  - Can handle different cases of the problem?

Why problems?



## Problems

- Problem transformation
  - From more realistic to more manageable
- Problem types

e.g., use of computers

  - Practical problems: Calls for an *action* (in reality)
  - Research problems: Calls for *information*
  - Computational problems: Calls for *computation*



## Research Problems

- As questions
  - *Yes-no* question: "Is Furby male?"
  - *Wh*-question: "What is the sex of Furby?"
- Cost *or* significance
  - Suffering of not being able to answer *or* benefit of answering [opposite sides]
  - Often, in connection to a practical problem
- Example: "How do ants find food?"

"The Craft of Research" (see the topic list)



## Digression: Analogy

- Problem (question) and significance
- Course (materials) and significance
- Organization and role
- Anatomy and physiology

Feeding problems to a computer

Preliminary

## Computational Problems

- Research problem 1: "Is Furby male?"
  - Computational problem (à la Prolog)
    - Input: **`furbySex(male).`**
    - Output: e.g., **`no.`**

Suppose the computer knows the answer

- Research problem 2: "What is the sex of Furby?"
  - Computational problem (à la Prolog)
    - Input: **`furbySex(X).`**
    - Output: e.g., **`X = female.`**



## Imagine a Lazy Machine

- Only responds with "uh" or "nah"
- Question: Can we phrase various problems in a way even the lazy machine can still be useful? If so, how?

Possibility of dealing with problems in a more uniform manner



## Converting Questions

- Any question (including *wh*-questions) can be converted into a series of *yes-no* questions.
  - *Wh*-question: "What is the sex of Furby?"
  - *Yes-no* questions: "Is Furby male?" "Is Furby female?"
- This technique allows us to deal with complex problems with "lazy" machines.



## Group Exercise 1

- Convert the following *wh*-questions into a series of *yes-no* questions
  1. What are the planets (of the Sun)?
  2. Where did you eat?
  3. When will the last human be born?

Representing formally?



## Formal Representation

- *wh*: "What is the square of $x$?" — Function
  - $f = \{(1, 1), (2, 4), (3, 9), ...\}$
  - I.e., $f(1) = 1$, $f(2) = 4$, $f(3) = 9$, ...
  - Pairs of an input and the output
- *yes-no*: "Is $y$ the square of $x$?" — Relation
  - $R = \{(1, 1), (2, 4), (3, 9), ...\}$
  - I.e., $(1, 1) \in R$, $(2, 4) \in R$, $(3, 9) \in R$, ...
  - Pairs of two inputs

Discrete math: review if necessary



## Number of Arguments

- 2 arguments
  - E.g., "Is 1 the square of 1?" "Is 4 the square of 2?" etc.
  - Cf. "What is the square of what?"
  - $\{(1, 1), (2, 4), (3, 9), ...\} = \{(x, y) \mid y = x^2\}$
- 1 argument
  - E.g., "Is 1 a square?" "Is 4 a square?" etc.
  - Cf. "What are squares?"
  - $\{1, 4, 9, ...\} = \{x \mid x \text{ is a square of some number}\}$

## Computational Problem

- Checking the membership of a set (could be a relation)
- Then, we may just identify a computational problem with the *set*. ~ *language* Recall this?
- Examples (cf. Group Exercise 1)
  - {*Mercury*, *Venus*, *Earth*, *Mars*, *Jupiter*, ... }
  - {(*Nobo*, *Penang*), (*Furby*, *home*), ...}
  - {2005, 2006, 2007, 2008, 2009, 2010, ...}

Potential problems with this description?



## Interim Summary: Problem Transformation

- Practical problem ~ action
  - Significance: Real-life situation
- Research problem ~ question
  - Significance: possibility of solving the associated practical problem (if acted)
- Computational problem ~ set
  - Significance: Abstract/computational treatment ⇒ Only yes-no response needed



## Practice: "Dating"

- Practical problem: Date an ideal partner.
- Research problem:
  - Cost/significance
- Computational problem (set):



## Group Exercise 2

- Transform the following research problems into the corresponding computational problems, and represent them as sets
  1. Is it going to rain this evening?
  2. Why do people (still) smoke cigarette?
  3. How can one cook pasta?

If you need clarification, ask questions



## Group Exercise 3

- Identify each of your own problem (Ex 00) with respect to the following stages:
  - Practical
  - Research
  - Computational
  - Set
- Transform your problems so that you eventually convert them into sets (may need to make adjustments)

Work on your problems **jointly within your group**



## Unit Summary

- Types of problems: practical, research, computational
  - Why categorize?
- Computational problem ~ set
  - Why set?
- Problem transformation
  - How to do it?

Cf. Content Goal 1

# Exercise A1: Using a Theory

- Part 1: Using a Theory
  - Task 1: Identify theories (in/out CS)
  - Task 2: Define “theory” (your own)
  - Task 3: Personal feelings about theory
- Part 2: Review “Problems”
  - Group Exercise 3 (previous slide)
- Unit A1 Summary question
  - Were you sufficiently motivated to discuss “problems” as the starting point of this course? Explain.

Name: ______________________

# Exercise A1, 1/21/05

## Part 1: Using a Theory

This is a course about Theory behind many computing concepts. However, the notion of "theory" is not necessarily well understood or appreciated. Before we can discuss components of the Theory of Computation, we had better confirm our understanding of theory in general.

**Task 1**: Identify multiple examples of theory (*both* in *and* outside computer science). Then, discuss how they can be used in practical situations, and comment on their usefulness.

**Task 2**: Give your own definition of "theory." Do not look up any reference. Be flexible and creative.

**Task 3**: Describe your personal feelings about theory. Were you always comfortable and content? Were there any time you felt that some theory is too abstract and/or useless?

Be prepared to discuss this part in class.

## Part 2: Review "Problems"

A lot of our activities (computational or not) begin with a problem. Thus, a good grasp of problems at different levels for appropriate processing is an important skill.

**Task**: Concisely write up your response to Unit A1 Group Exercise 3 (in class; slides available on-line). That is, re-do the exercise for your own problem from Exercise 00. Certain problems may not be neatly analyzed in the way discussed in class. If your problem falls in that category, explain why.

Note: If you think you understood how to do this part, you should place a check mark for the criteria in Content Goal 1 (eval form). Then, you can fill in your supporting notes with a brief description of how you were able to achieve those criteria. If you have questions, contact the instructor right away.

Instructions/Notes (for both parts):

1. Follow the general guidelines given in Exercise 00. ................................ ☐
2. Clearly identify parts and tasks. ................................ ☐
3. Review the evaluation form and start to write your supporting notes. ................ ☐

Survey: Time spent between classes: ______________

// End

# Before Starting

- Questions about Ex A1 Part 2 (Review)
- A practical problem of the past weekend: show shoveling
  - Research problem?
  - Computational problem (set)?
- Why theory?



# Unit A2: Overview

- Discuss your experience with "theory"
- Explore notions of theory
- Discuss important properties of theory
- Preview Exercise A2 "Your Ideas about Theory of Computation"

Use of your problems in this course



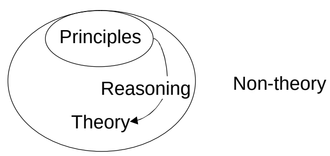
# Exercise A1 Part 1

- Part 1: Using a Theory
  - Task 1: Identify theories (in/out CS)
  - Task 2: Define "theory" (your own)
  - Task 3: Personal feelings about theory



# Theory

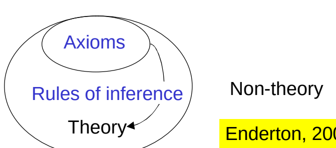
- Consistent, i.e., no contradiction
- Potentially infinite amount ~ predictability
- Principled, i.e., derivable from a small number of principles via reasoning






# Theory (in Logic)

- Consistent, i.e., no contradiction
- Potentially infinite amount ~ predictability
- Axiomatized, i.e., derivable from a small number of axioms via rules of inference




Enderton, 2001 (see the topic list)



Optional

# Theory, More Formally

- **Language**: Specification of the use of symbols.
  - **Formula** (well-formed formula): Syntactically correct sequence of symbols that would evaluate either true or false.
- **Interpretation**: Specification of the meaning of symbols under consideration.
- **Structure**: Specification of the meaning of symbols *not* defined by the interpretation.
- **Satisfaction**: A structure $\mathbf{A}$ satisfies a formula $\varphi$ if $\varphi$ is true with respect to $\mathbf{A}$. This relation is usually written as $\mathbf{A} \models \varphi$.
- **Logic**: Combined specifications of language, interpretation, and satisfaction.
- **Model**: For a logic $L$, a structure $\mathbf{A}$ is called a model of formulas $\Phi$, if $\mathbf{A} \models \varphi$ for all $\varphi \in \Phi$. The collection of models of $\Phi$ is written as $\mathrm{Mod}(\Phi)$.
- **Logical consequence**: For a logic $L$, a formula $\varphi$ is called a logical consequence of $\Phi$, if for each $\mathbf{A} \in \mathrm{Mod}(\Phi)$, $A \models \varphi$.
- **Theory**: For a logic $L$, a theory is a set of formulas $\Phi$ such that for each formula $\varphi$, $\Phi \models \varphi$ implies $\varphi \in \Phi$. [In practice, we also require consistency.]

Supplemental notes available on-line

## Are These Theories Useful?

- Example 1
  - Axioms: Newton's law of physics
  - Rule of inference: Scientific reasoning
  - Theory: Classical physics
- Example 2
  - Axiom: "0" is a number.
  - Rule of inference: If $n$ is a number, so is $n + 1$.
  - Theory: The set of natural numbers



## Significance of a Theory

- The notion of "theory" (at least in a technical sense) simply identifies a way some knowledge is organized.
- The usefulness of a theory depends on factors outside the theory. For example, the axioms and rules of inference must "reflect" our observation and intuition.
  - Cf. problem and its cost



## Digression: Analogy

- Theory and significance
- Logic and (mathematical) structure
- Syntax (sentence organization) and semantics (meaning)
- Form and content

Recall Unit A1
- Problem (question) and significance
- Course (materials) and significance
- Organization and function
- Anatomy and physiology



## Practice: "Good" Theory

- Discuss the following criteria proposed to identify a good theory, with respect to our characterization of theory.
  1. Reflect the real world?
  2. Supported by convincing evidence?
  3. Explain the past and predict future outcomes?
  4. Handle new data and discoveries?
  5. Clearly understandable, simplify rather than complicate the world?



## Caveat: Axioms

- For a theory to be useful, axioms must be "appropriate" with respect to our experience and/or intuition.
- However, axioms are still assumptions. In general, there is no way to logically justify their correctness.

Scientific theory as a big "if" statement



## Interim Summary: Theory

- Form
  - Axioms, rules of inference, theory
- Content
  - Grounding (reflecting observation/intuition)
  - Significance (usefulness)
  - Nature of axioms
  - Properties: consistency, predictability, efficiency, etc.

## Practice: Blood Vessels

- Schematic representations
- Comparison with other systems
- Involved properties
- Developing a theory for analogous phenomena
  - Axioms:
  - Rule of inference:
  - Theory:



Preparation for Group Exercise 1

## Anything in Common?

1. Word frequency (in a text)
2. City size
3. Intensity of wars
4. Web page links
5. Forest fires
6. Earthquakes
7. Heartbeat



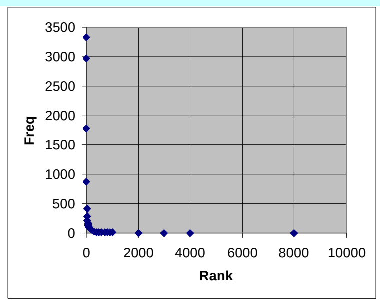
## Word Frequency in *Tom Sawyer*

| Word | Freq | Rank | F x R |
|---|---|---|---|
| *the* | 3332 | 1 | 3332 |
| *and* | 2972 | 2 | 5944 |
| *and* | 1775 | 3 | 5325 |
| *he* | 877 | 10 | 8770 |
| *but* | 410 | 20 | 8200 |
| *be* | 294 | 30 | 8820 |
| *there* | 222 | 40 | 8880 |
| *one* | 172 | 50 | 8600 |
| *about* | 158 | 60 | 9480 |
| *more* | 138 | 70 | 9660 |
| *never* | 124 | 80 | 9920 |
| *Oh* | 116 | 90 | 10440 |
| *two* | 104 | 100 | 10400 |

| Word | Freq | Rank | F x R |
|---|---|---|---|
| *turned* | 51 | 200 | 10200 |
| *you'll* | 30 | 300 | 9000 |
| *name* | 21 | 400 | 8400 |
| *comes* | 16 | 500 | 8000 |
| *group* | 13 | 600 | 7800 |
| *lead* | 11 | 700 | 7700 |
| *friends* | 10 | 800 | 8000 |
| *begin* | 9 | 900 | 8100 |
| *family* | 8 | 1000 | 8000 |
| *brushed* | 4 | 2000 | 8000 |
| *sins* | 2 | 3000 | 6000 |
| *Could* | 2 | 4000 | 8000 |
| *Applausive* | 1 | 8000 | 8000 |

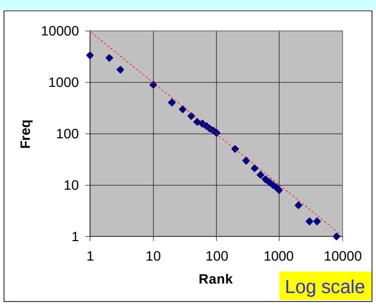


## Graphical Representation (1)






## Graphical Representation (2)



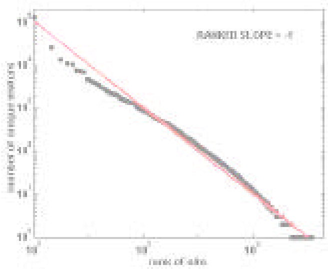



## Power Law

For all the mentioned phenomena

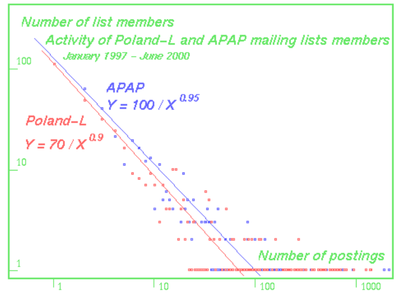
- (roughly) The scale-frequency relation as a straight line with the slope –1 on a log-log scale graph.
- (formally) $Freq \propto Scale^{-1}$

## Web Site Visits



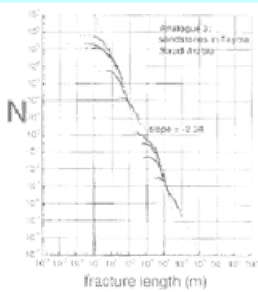


## Mail List Activities



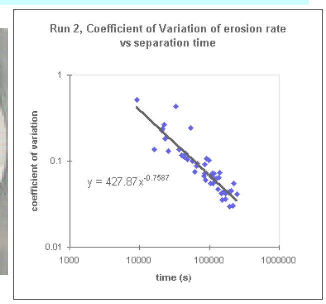



## Material Fracture Pattern



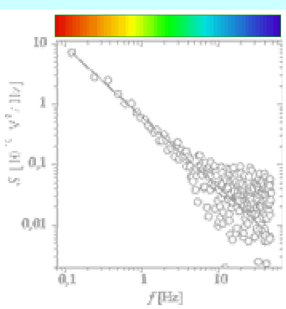



## Shore Erosion






## “Pleasant” Musical Tones






## Group Exercise 1

- Examine and evaluate the following theory (with respect to consistency, predictability, significance, etc.)
  - Axioms: Many real-world phenomena (natural and artificial) “scale” (i.e., small ↔ large).
  - Rule of inference: If a phenomenon scales *x* times, such a phenomenon is *x* times unlikely to occur.
  - Theory: Many real-world phenomena follow the “power law” ($Freq \propto Scale^{-1}$).

## Group Exercise 2

- Develop a theory of "justice"
  - Axioms:
  - Rules of inference:
  - Theory:
- Examine the following properties
  - Consistency
  - Predictability
  - Usefulness



## Group Exercise 3

- What kind of theory would be useful to analyze/solve your own problems?
- Speculate the form of the theory (axioms, rules of inference, theory).
- Examine the following properties
  - Consistency
  - Predictability
  - Usefulness

Work on your problems
**jointly within your group**



## Unit Summary

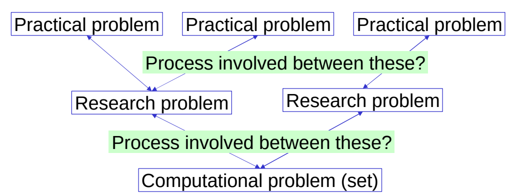
- Theory: Axioms, rules of inference, theory
- Significance of a theory
- Nature of axioms
- Analyzing theory

Problems and theory?



Preview

## Problems and Theory




- What can we do after this?
- Where do theories apply?



## Exercise A2: Theory of Computation

- Your ideas about Theory of Computation
  - Task 1 (review): Identify/analyze CS "theories" you know
  - Task 2 (~ Group Exercise 3): Usefulness of the theories you know; missing elements
- Unit A2 Summary questions
  - Do you have a better idea about "theory" after this meeting? Explain.
  - Questions/Comments/Suggestions

Name: ______________________

# Exercise A2, 1/25/05

## Your Ideas about Theory of Computation

In this course, you are expected to improve and refine your understanding of the Theory of Computation so that you can *apply* it to your problems that would surface in your career (life). In this part of the exercise, you will examine your current understanding related to the Theory of Computation, as a starting point. Note that you are not expected to "study" to respond to this exercise. Write as much as you know at this point.

**Task 1** (review): Identify ideas/concepts you know (in Computer Science) that can be considered as theories. Try to analyze *both* the organization (i.e., axioms, rules of inference, and theory) *and* the significance of each of the examples. Although many ideas discussed in CS may not have been labeled as "theories," there are many concepts that can be considered theory. If you have difficulty coming up with a candidate, identify *some* theory (not necessarily in CS, e.g., Unit A2 Group Exercise 2 "Justice") and do the same.

**Task 2** (in connection to Unit A2 Group Exercise 3): Examine whether any of the theories you identified in **Task 1** is useful for analyzing/solving your own problems (e.g., those in Exercise 00). If you think the theories you know are not sufficient to tackle your problems, what kind of theory would you need?

Be prepared to discuss this part in class.

Instructions/Notes:

1. Follow the general guidelines given in earlier exercises. .................................................. □

Survey: Time spent between classes: ______________

// End

# Unit A3 Supplement, 1/29/05

Here is my response to your summary questions (rephrased/combined in some case).

**Q1**: Possible to express the "actual" solution of a problem in set notation, esp. when the number of possible solution is very large? How general the predicate notation on a set could be.

**A1**: The list notation is good only for a finite set. But as long as the set is finite, you can still contain a large number of members, at least theoretically. Many problems have infinitely many solutions. The "problem" (or solution) set for such problems must be in the predicate notation. If the specification part (at the right of the bar '|') of a representation is precise enough for the "processor" to interpret, the predicate notation is good enough. For example, $\{(c, e) \mid c \text{ causes } e\}$ would be good only for informal discussion, not for a computational process, while $\{(x, y) \mid x < y, \text{ where } x, y \in \mathit{NaturalNumbers}\}$ can be processed mechanically.

**Q2**: Representation of sets besides the basics?

**A2**: When you deal with a real-world problem, the data can be complex. As an example, let us consider Group Exercise 3 (Slide 24), which we did not have time to discuss. Here is the first attempt: {(*course*, *instructor*) | *instructor* teaches *course*}. Would this be right? No, because it characterizes a single instance of course-instructor mapping. The problem must address the entire map that we get the complete information about who is teaching what in within realistic constraints. How about this: {(*courses*, *instructors*, *assignment*) | *assignment* is an injective function from *courses* to *instructors*, where *courses* and *instructors* are non-empty sets of available courses and instructors}? For example, imagine *courses* = {410, 460, 470}, *instructors* = {NN, NK, PD, MM}, and *assignment* = {(410, PD), (460, MM), (470, NN)}. *assignment* is a function (why?) and injective (why?). Now, you should be able to see how the materials discussed in Discrete Structures begin to play a role. [Review questions: What's wrong if *assignment* is not a function or injective? Why doesn't *assignment* need to be surjective?]

**Q3**: Problems in CS for which it is impossible to prove whether they are "computable" or not? How to finally decide if something is "computable" or not?

**A3**: To answer these questions, we need to know a little more about the notion of "computability," which we will discuss in the rest of Module A and much of Module B. Keep your question. As a side note, I want to mention a logical problem that cannot be proven true or false. Consider the sentence: "I'm telling a lie." Am I really telling a lie? This "liar paradox" has a close connection to "computability." We will probably touch on this point later.

**Q4**: The distinction between what is necessarily included in the discussion about complexity vs. what is excluded?

**A4**: The complexity area of the Theory of Computation focuses on the time or space performance of algorithms as a function of their input size. In this area, we only discuss algorithms, which must terminate for all inputs; thus, computability is not an issue. In addition, we only deal with mechanisms that are capable of solving the problem in hand; thus, choosing the simplest mechanism is not an issue.

**Q5**: How to apply/figure out the properties in the three subareas of the Theory in "my" problems or other problems like the floor tiling?

**A5**: The entire course is supposed to be the opportunity for you to gain this ability. So, at this point, you should not worry about not being able to do this as much as you wish. However, I hope you got some sense of the three subareas of the Theory and be able to guess which areas are more relevant to your own problems. To get some additional ideas, you may want to use the sample problems from Spring 2003 (http://www.tcnj.edu/~komagata/csc460/05s/SampleProblems.pdf). Problems 1 through 10 primarily involve issues in computability; Problems 11 through 19 primarily involve issues in complexity; Problems 20 through 23 primarily involve issues in languages/automata. As I mentioned in class, most of your problems have some relevance to some of these areas. In addition, if your problem calls for a program that should not terminate or involves high degree of interactivity, it might require analysis beyond the traditional Theory. For example, the robotic control of automobile must not terminate and my Birds program is most appropriately designed as interaction of autonomous birds.

**Q6**: How can formal models of interactivity be mathematically expressed?

**A6**: I am planning to include a very basic discussion related to this topic near the end of the semester. Some references are listed in the section, "Beyond Turing computability" of the references file (http://www.tcnj.edu/~komagata/csc460/05s/References.pdf). Here is a preview of the discussion. To address interactivity (or computation beyond algorithms), people have proposed different approaches:

- Instead of discrete data/processing, use analog computation (e.g., analog version of artificial neural network)
- Instead of limiting to a single input-processing-output session, allow the mechanism to repeat the process with some memory between sessions
- Instead of prohibiting interaction within the process, allow the process to interact with (or consult) another process.

With any of these approaches, we can show that it would become possible to do what was not possible with a single session of algorithmic computation.

If you have more questions, let me know.

// End

# Before Starting

- Questions
- Computational problems as sets
  - Need to know the answer in advance?
  - Representing the relation between inputs and the problem schematically?



# Unit A3: Overview

- Discuss your ideas about “Theory of Computation”
- Preview the main components of the Theory of Computation
- Practice representing computational problems as sets
- Preview Exercise A3 “Computational problem solving”

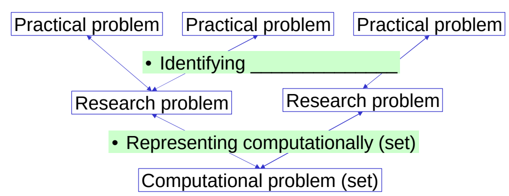


# Exercise A1 Part 1

- Your ideas about Theory of Computation
  - Task 1 (review): Identify/analyze CS “theories” you know
  - Task 2 (~ Group Exercise 3): Usefulness of the theories you know; missing elements



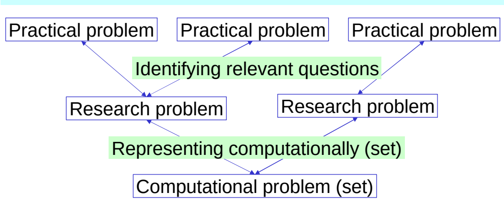
# Problems and Theory




Properties to analyze (about the problem)?



# Problems and Theory



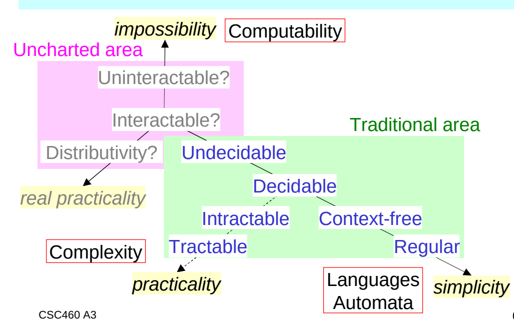

- Possible to compute at all?
- How to choose the simplest mechanism?
- Practical for any future inputs?

**Theory**

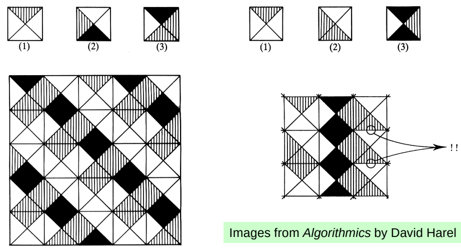
Significance?



# Preview: Theory of Computation



## Example 1: Floor Tiling




Images from *Algorithmics* by David Harel



Cf. some of your own problems



## Computability

- Content Goal 4
- Example questions
  - What are the limitations of computation? All the computational problems (sets) solvable?
  - How can we compare different forms of “computation?”
  - What is the notion of “computation?”
- Significance?
- Are your own problems “computable?”



Schematic representation



## Example 2: Survivor

- Scenario
  - Alone on a large island where you survived an aircraft crash.
  - Established a base where you can spend your nights safely.
  - Still need to explore the island to obtain foods.
- Appropriate (minimal) mechanism for your “mental” computer?



Cf. some of your own problems



## Languages/Automata

- Content Goal 5
- Example questions
  - Where are the balance between the simplicity of computational mechanisms and their abilities?
  - How can we identify an appropriate computational mechanism for a given problem (set).
- Significance?
- The simplest mechanism for your own problems?



## Example 3: Scheduling

- How to define the problem?
- Must be sufficiently general to be able to handle cases including:
  - Your time management
  - Organizing a team of workers
  - Assigning courses to instructors
  - Schedule the analysis process for the entire human genome



Cf. some of your own problems



## Complexity

- Content Goal 6
- Example questions
  - How does the input data size affect the computation?
  - What would be the limit of “practical” computation?
- Significance?
- Practicality of your own problems, esp. facing large data?

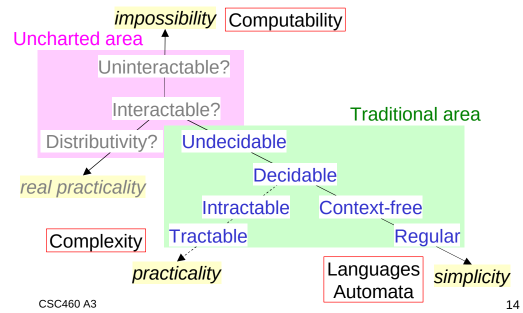
## What is not usually discussed...

- Content Goal 3: Interactivity
- Example questions
  - How well does algorithmic computation fare in real-world computation?
  - What are the essential properties that are needed to solve real-world problems computationally?
- Significance? Examples? Implications?

These questions as computational problems?



## Preview: Theory of Computation






## Interim Summary

- Theory of Computation
  - Offers principles in three main aspects: computability, languages/automata, complexity, most commonly dealing with problems as sets
  - Limitations due to its traditional foundation on algorithmic computation, cf. interactivity
- Caveat
  - Focus on computational problems (sets), i.e., no systematic methods for problem transformation, cf. discrete math



## Practice: Robotics

- Suppose that robotic control of an automobile has been transformed into a computational problem (set).
- Analyze with respect to the following:
  - Computability (possibility)
  - Languages/Automata (simplest mechanism)
  - Complexity (practicality with respect to the input size)



## Group Exercise 1

- Suppose that a typical compilation problem has been transformed into a computational problem (set).
- Analyze with respect to the following:
  - Computability (possibility)
  - Languages/Automata (simplest mechanism)
  - Complexity (practicality with respect to the input size)



## Set Representation of Problems

- Benefits
  - To analyze/compute such problems, we only need a single mechanism to check set membership
  - Well-developed techniques in mathematics are available
- Limitations?

## Alternative Forms

- Computational problem
- Set
- Language
  - Set of strings (special case of set)
- Characteristic function
  - Outputs yes/no, i.e., comparable to a relation (set of tuples)



## Set Examples

1. What is the sex of Furby?

   {*female*}
2. What are palindromes?

   {*a*, *cc*, *wow*, *abba*, *kayak*, *hannah*, ...}
3. Addition operation (on natural numbers)

   {(0, 0, 0), (0, 1, 1), (1, 0, 1), (1, 1, 2), ...}
4. Causal relation (on any possible event)

   {(*drink*, *joy*), (*drink*, *nausea*), (*drink*, *addiction*), ...}

Not knowing answers; Avoiding "..."



## Set Notation (Review)

- List notation (technically, only for finite sets)
  - {1, 2, 3, 4, 5}
  - {(paper, stone), (scissors, paper), (stone, scissors)}
- Predicate notation
  - $\{x \mid 0 < x \leq 5, x \text{ is a natural number}\}$
  - $\{(x, y) \mid x \text{ wins over } y \text{ in the paper-scissors-stone game}\}$
  - $\{x \mid x \text{ is a number never used by the human}\}$

Predicate notation as a filter



## Practice: Predicate Notation

- Palindromes

  {a, cc, wow, abba, kayak, hannah, ...}

  = $\{x \mid x \text{ is a palindrome}\}$ A better way?

  = { }
- Addition operation (on natural numbers)

  {(0, 0, 0), (0, 1, 1), (1, 0, 1), (1, 1, 2), ...}

  = { }
- Causal relation (on any possible event)

  {(drink, joy), (drink, nausea), (drink, addiction), ...}

  = { }



## Group Exercise 2

- Infinite loop is one of the most common bugs in any program. It would be enormously helpful if someone writes a program to detect infinite loops in a given program. **Give** the set representation of the computational problem involved here. **Speculate** the basic Theory properties (i.e., possibility, etc.), referring to the set (a concise, informal description suffices).



## Group Exercise 3

- For any (possibly very large) academic department, how can we assign all the courses to the instructors within the usual constraints (e.g., a single instructor for a single course)? **Give** the set representation of the computational problem involved here. **Speculate** the basic Theory properties, referring to the set.

# Unit Summary

- Theory of Computation: computablity, languages/automata, complexity
- Set representation of computational problems: list/predicate notations
- Understand Unit A3 Exercise
- Summary question
  - We discussed broad ideas with little details. So, you must have questions or be uncertain on at least some aspect. What are they?

Name: ______________________

# Exercise A3, 1/28/05

## Part 1: Computational Problem Solving

As we glanced at the main components of the Theory of Computation, we will be ready to discuss the components more in detail. The standard interface between your practical problems and the theoretical analysis is the set representation of your (corresponding) computational problem.

**Task 1 (Program Verification)**: One of the software industry's biggest concern is how to check whether their products are really correct with respect to the specification that the developer and the user both agreed. According to a variety of sources, inability to do so will cost us an incredible amount of money in the future (well, this must be already happening). As you know, *generality* is a prime concern in Computer Science. So, why don't we write a single program that could verify whether the program correctly solves a problem with respect to a given specification? **Give** the set representation of the computational problem involved here. **Speculate** the basic Theory properties (i.e., possibility to solve computationally, the simplest mechanism, practicality with large data), referring to the set (a concise, informal description suffices). Can you think of such a program?

**Task 2 (Map Coloring)**: It has been shown that with four distinct colors, we can color any map so that the neighboring countries (or whatever political boundaries) do not share the same color. With three colors, we may or may not be able to do the same thing. **Give** the set representation of the computational problem involved here. **Speculate** the basic Theory properties, referring to the set.

## Part 2: Review "Theory of Computation"

Without going into the details of the Theory of Computation, we discussed the properties associated with the three traditional subareas of the Theory. While it must be difficult to point out exactly how these would apply to your problems, you should still be able to *speculate* how these property would apply (intuition is important!).

**Task**: Suppose that your own problem (Ex 00) has been transformed into a computational problem (set). Concisely analyze the problem with respect to the basic Theory properties. If you have difficulty, explain where you have the difficulty. You are also encouraged to discuss with other students and/or the instructor.

## Part 3: Evaluation Form and Supporting Notes

Module A evaluation will come in a week or so. You must be continuing to fill in the evaluation form and writing up your supporting notes as much as you can. You are also encouraged (but not required; i.e., not required to submit anything for this part) to attach a copy of your supporting notes to this exercise so that the instructor can comment on them.

Survey: Time spent between classes: ______________

// End

## Questions/Review

- Ex A3
  - Part 1: Computational Problem Solving
    - Task 1: Program verification
    - Task 2: Map coloring
  - Part 2: Review
  - Part 3: Evaluation form and supporting notes
- Schematic representation
  - Set representation of a problem
  - Analyzing the set regarding Theory properties

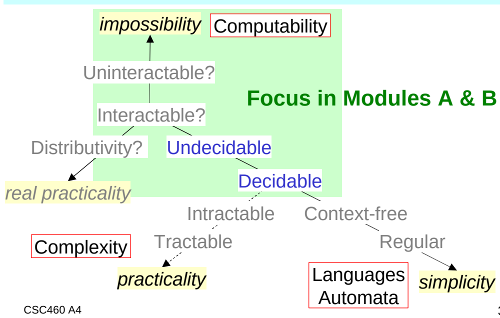


## Unit A4: Overview

- Overview: “Computability”
  - Intuition first; reasoning later (Module B)
- Introduce Turing machines (TMs)
  - Definition, properties, behaviors
- Learn how to operate TMs
- Preview Exercise A4 “Test-Drive TMs”
  - Using the JFLAP simulator



## Overview: Theory of Computation





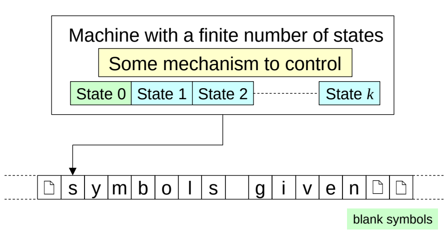

## Tools for Computability Analysis

- Need for a standard abstract mechanism of computing sets
- Some possibilities
  - Some modern programming language
  - Some assembly language
  - Some kind of pseudo code
  - Mathematical functions
  - Other



## Turing Machine (TM)

- Proposed by Alan Turing (1936)

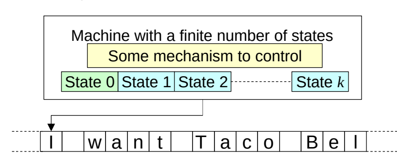



Digression: more about Turing



## Example Processes

- Word-by-word translation from English to Spanish, e.g., “I want Taco Bell”
- Detect palindromes
- Detect binary numbers



## Additional Properties

- The tape has no left or right end.
  - Tape positions can be read/written.
  - The tape head can move left/right.
  - A tape position may be blank (□). The positions outside the input are filled with blanks.
  - The initial tape position is the leftmost input position.
- There are finite number of states.
  - There is a unique start state.
  - There are designated accept states.
- The input/tape alphabet contains a finite number of symbols.



## TM Input

- Any string (of a designated alphabet)
- One instance of the complete information to be tested for its acceptability (cf. yes-no)
- Example
  - For the problem $\{(x, y, z) \mid x + y = z, \text{ where } x, y, z \in \mathit{NaturalNumbers}\}$, the following can be given as an input "...□1#2#3□...". A reasonable TM would accept this input. Cf. the binary representation: "...□1#10#11□..."



## Possible TM Behaviors

- Halts
  - Accepts: If in one of the accept states
  - Dies: Otherwise
- Loops

Note: Assume no looping from an accepting state

- Information left on the tape
  - Often, ignored (side effects)
  - Possible to interpret it as the output



## TM's Yes-No Response

- Option 1 (ternary)
  - Yes: If accepts (of course, halting)
  - No: If dies (still halting)
  - Don't know: If looping
- Option 2 (binary)
  - Yes: If accepts (of course, halting)
  - Not yes: Otherwise (i.e., no distinction between dying vs. looping)

Note: Possibly no response



## TM Variations

- Single accept state
- Both accept/reject states
- Semi-infinite tape
  - I.e., the existence of the left end
- Multiple tapes
- Nondeterministic transition
  - I.e., $\delta$ as a relation, not a function

Cf. Sipser 1997: Semi-infinite tape, single accept, single reject



## TMs vs. Computers

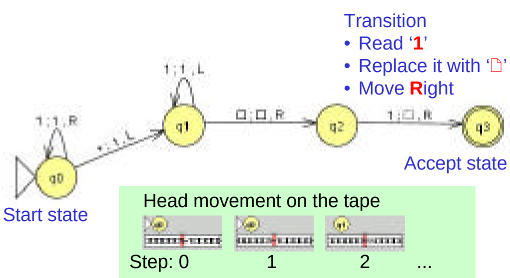
- Similarities and differences?

## Interim Summary

- TMs are a simple, intuitive, and precise model of computation.
- The behavior of a TM is to capture the notion of "computation," also referred to as "effective procedure."
- When an effective procedure is guaranteed to terminate, it is called "algorithm."



## Transition Diagram (JFLAP)






## Formal Definition

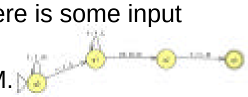
TM $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$

- $Q$: finite set of states
- $\Sigma$: finite set of input symbols, $\square \notin \Sigma$ (blank symbol)
- $\Gamma$: finite set of tape symbols, $\square \in \Gamma$, $\Sigma \subseteq \Gamma$
- $\delta$: transition function $Q \times \Gamma \rightarrow Q \times \Gamma \times \{Left, Right\}$
  - E.g., $\{((q_1, 0), (q_2, 1, R)), ((q_2, 1), (q_3, \square, L)), ...\}$
- $q_0$: the start state $q_0 \in Q$
- $\square$: blank symbol ($B$ in the text)
- $F$: set of accept states $F \subseteq Q$



## Practice: Detecting Input

- Draw a transition diagram of a TM that would accept when there is some input (limit to 0 or 1).
- Formally define the TM.



TM $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$

- $Q$: finite set of states
- $\Sigma$: finite set of input symbols, $\square \notin \Sigma$
- $\Gamma$: finite set of tape symbols, $\square \in \Gamma$, $\Sigma \subseteq \Gamma$
- $\delta$: transition function $Q \times \Gamma \rightarrow Q \times \Gamma \times \{Left, Right\}$
- $q_0$: the start state $q_0 \in Q$
- $\square$: blank symbol ($B$ in the text)
- $F$: set of accept states $F \subseteq Q$



## "Language" Examples

- Any sequence of 0's and 1's with exactly one '#': $\{x\#y \mid x, y \in \{\mathbf{0}, \mathbf{1}\}^*\}$
- Any sequence of 0's and 1's repeated after exactly one '#': $\{w\#w \mid w \in \{\mathbf{0}, \mathbf{1}\}^+\}$
- Any distinct sequences of 0's and 1's delimited by '#': $\{\#x_1\#x_2...\#x_l \mid \text{each } x_i \in \{\mathbf{0}, \mathbf{1}\}^+ \text{ and } x_i \neq x_j \text{ for each } i \neq j\}$
- Any sequence of the same number of 0's and 1's in that order: $\{0^n1^n \mid n > 0\}$



## Unit Summary

- Understand the basics of Turing machines
  - Description, properties, behaviors
  - Transition diagram, formal definition, JFLAP
- Start Exercise A4 "Test-Drive TMs" in groups, in the lab
- Summary question
  - If TMs did not exist, what would you use as a model of computation? Explain.
  - Questions/Comments/Suggestions

# Unit A4: Another Binary Adder Example, 2/6/05

After reviewing your TM binary adders, I came up with my own version. The main idea is similar to Jared's multiple-tape TM (but without using multiple tapes). I also borrowed Eric's idea of using the little endian format (but with the input formatting following the exercise sheet, i.e., 1+01=11 for 1+2=3, without using additional tape symbols for "carry"). If one still wants to get input in the big endian, it would be possible to add a preprocessor to convert the format, using Scott's binary number reverser.

Here is the explanation of the main mechanism. The TM would compare the leftmost bit of each segment. In the following example, there is nothing wrong with the part "0 + 1 = 1":

...□01+11=101□□...

If it were "0 + 1 = 0", the TM can immediately tell the input is not acceptable, by terminating there. The TM will also need to record whether there will be carry. This can be done by using a distinct set of states. Thus, the TM distinguishes the following four acceptable cases: "0 + 0 = 0", "0 + 1 = 1", "1 + 0 = 1", "1 + 1 = 0" (with carry). Since the only information the TM needs for the next round is the presence/absence of carry, the processed bits can be replaced. Using the following replacement pattern, the tape configuration becomes analogous to the original configuration (except that there are repetitive delimiters, '+' and '=').

...□□1++1==01□□...

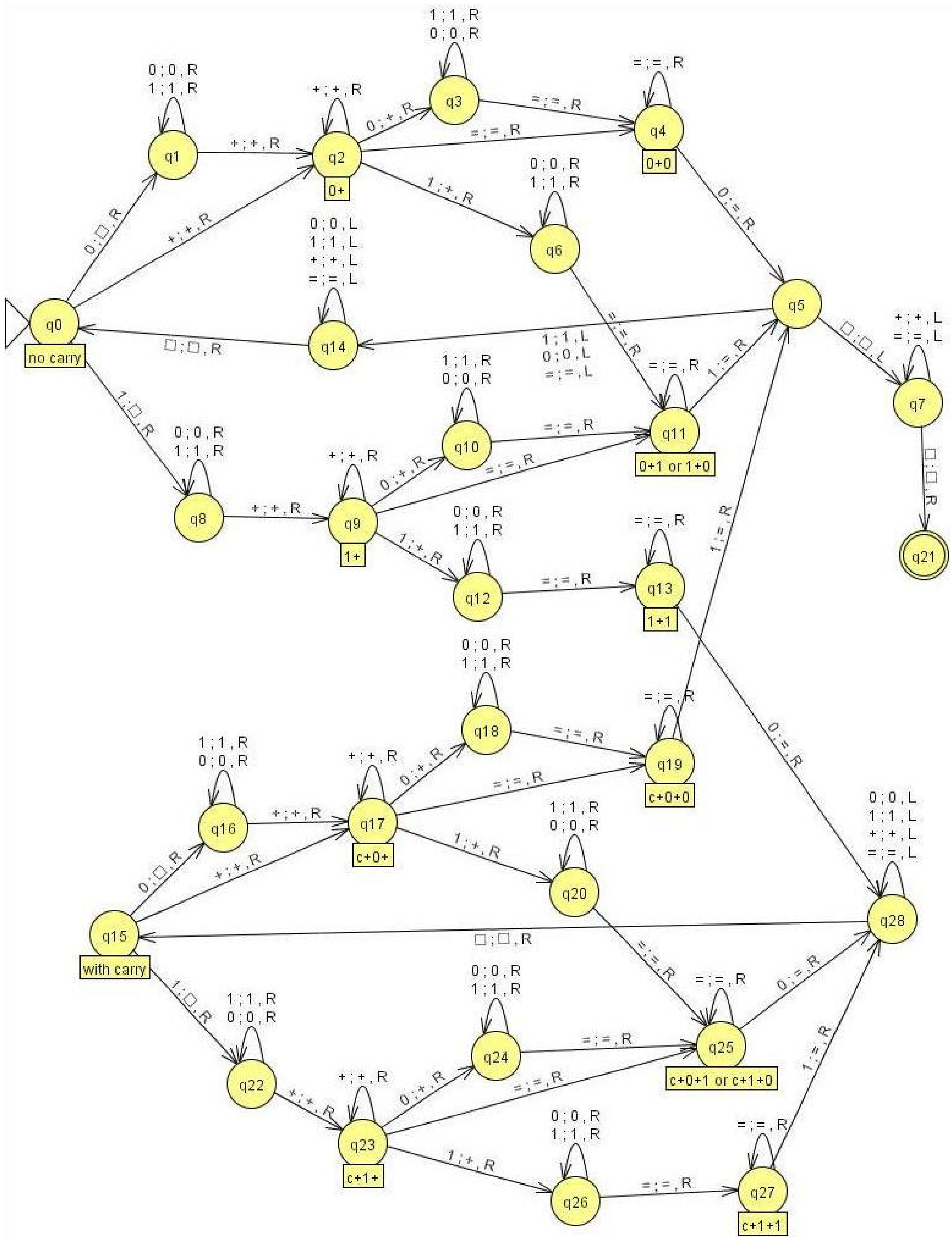
Then, we have an analogous, but smaller problem (cf. divide and conquer). The current focus "1 + 1 = 0" is correct if we process the carry bit in the next around. This TM contains two similar submachines: one for the case with no carry and the other for the one with carry. As done in Eric's machine, it is possible to encode the distinction on the tape (with fewer states, but more complex tape operation). Since the argument bits are already consumed, the TM interprets missing bit as 0. Then, "carry + 0 + 0 = 1".

...□□□+++===1□□...

If the input survived this much, all the bits are consumed. The TM can then check for this by scanning only '+' and '='.

...□□□+++===□□...

The transition diagram is on the next page, and the code (nk-adder.jff) is available in our JFLAP folder (http://www.tcnj.edu/~komagata/csc460/05s/JFLAP/). One unusual aspect of this version is that the input "+=0" will also be accepted because missing arguments across '+' are interpreted as 0. I tested with various inputs, but I have no doubt you can come with problematic inputs. If you test-drive this TM and identify a bug (or be convinced that it is correct), you will receive some reward.


q0
no carry
q1
q2
0+
q3
q4
0+0
q5
q6
q7
q8
q9
1+
q10
q11
0+1 or 1+0
q12
q13
1+1
q14
q15
with carry
q16
q17
c+0+
q18
q19
c+0+0
q20
q21
q22
q23
c+1+
q24
q25
c+0+1 or c+1+0
q26
q27
c+1+1
q28

// End

# Unit A4 Supplement, 2/1/05

Here are sample response to Exercise A3 Part 1

**Task 1 (Program Verification)**: One of the software industry's biggest concern is how to check whether their products are really correct with respect to the specification that the developer and the user both agreed. According to a variety of sources, inability to do so will cost us an incredible amount of money in the future (well, this must be already happening). As you know, *generality* is a prime concern in Computer Science. So, why don't we write a single program that could verify whether the program correctly solves a problem with respect to a given specification? **Give** the set representation of the computational problem involved here. **Speculate** the basic Theory properties (i.e., possibility to solve computationally, the simplest mechanism, practicality with large data), referring to the set (a concise, informal description suffices). Can you think of such a program?

$\{(s, p)$ | Specification $s$ specifies program $p\}$
$\{(s, p, i)$ | Specification $s$ specifies program $p$ when run on input $i\}$
Note 1: It would be difficult to make the notion of "specification" more precise for this exercise.
Note 2: Including collections of specifications and programs in this representation would view the problem at a level higher than the original.

**Task 2 (Map Coloring)**: It has been shown that with four distinct colors, we can color any map so that the neighboring countries (or whatever political boundaries) do not share the same color. With three colors, we may or may not be able to do the same thing. **Give** the set representation of the computational problem involved here. **Speculate** the basic Theory properties, referring to the set.

$\{(R, C, N, a)$ |
$R$ is a non-empty set of regions,
$C$ is a non-empty set of colors where $|C| = k$ (for some positive integer $k$),
$N$ is a binary relation on $R$ specifying the neighboring regions,
$a$ is a function from $R$ to $C$ assigning a color to a region,
To specify different colors for two neighboring country: if $(x, y) \in N$, $a(x) \neq a(y)\}$

E.g., $\{(\{r_1, ..., r_n\}, \{c_1, ..., c_4\}, \{(r_1, r_2), (r_1, r_3), ...\}, \{(r_1, c_2), (r_2, c_3), ...\}), ...\}$

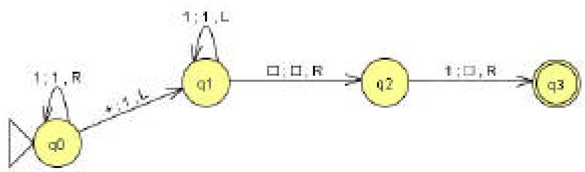
// End

Name: ______________________

# Exercise A4, 2/1/05

## Part 1: Test-Drive Turing Machines

Once your problem is represented as a set (precisely and discretely), you can analyze various computational properties in a formal manner. The standard tool to analyze "computability" (e.g., whether an algorithm exists for the problem) is the Turing machine (TM). By understanding this tool, we can access the rich literature and also make a lot of connections with other forms of analytical tools. So, in this exercise, you test-drive them, using a simulator called JFLAP (http://www.cs.duke.edu/~rodger/tools/jflap/; links are clickable on-line; also linked from the on-line syllabus). A copy of the program (`JFLAP.jar`) and some examples (`*.TM, *.TM.jff`) are available locally (http://www.tcnj.edu/~komagata/csc460/05s/JFLAP/).




TM $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$

- $Q$: finite set of states
- $\Sigma$: finite set of input symbols, $\square \notin \Sigma$
- $\Gamma$: finite set of tape symbols, $\square \in \Gamma$, $\Sigma \subseteq \Gamma$
- $\delta$: transition function $Q \times \Gamma \to Q \times \Gamma \times \{Left, Right\}$
- $q_0$: the start state $q_0 \in Q$
- $\square$: blank symbol ($B$ in the text)
- $F$: set of accept states $F \subseteq Q$

**Task 1**: Using JFLAP, design a TM that would loop regardless of the input. Choose one of the following options to show your work: (1) attach/copy/draw (the screen shot of) the transition diagram of your TM, or (2) include the formal definition of your TM. In addition, concisely explain how your TM would work.

**Task 2**: Consider a problem represented as $\{x+y=z \mid$ when $x$, $y$, and $z$ are interpreted as non-negative binary numbers in some standard representation, the arithmetic relation $x + y = z$ holds$\}$. In this representation, you may consider "$x+y=z$" as a variant form of $(x, y, z)$ with special delimiters. Repeat the process as in **Task 1**. Note that this task will be challenging, tedious, or both. You can stop when you feel you spent enough time.

**Task 3**: Define an interesting, non-trivial problem of your own *as a set*. Then, repeat the process as in the earlier **Tasks**. Again, you can stop when you feel you spent enough time.

## Part 2 (optional): TM Variations

Note: Depending on your time and interest, you decide whether to do this part.

If you look up references, you will occasionally find different versions of TMs. Initially, it might be confusing. However, if you think carefully, it is possible to see those variations as different ways of representing the same notion of "computation." But what does it mean to be the "same?" We will discuss this point more in detail in Module B. This part of the exercise is a preview and can be considered as an extended exercise on TMs.

**Task 1**: In another Theory of Computation textbook (Sipser, 1997), Turing machines are defined differently from our version on the following points:

1. Use of a semi-infinite tape. The input string must be placed at the left end of the tape.

2. There are exactly one accept state *and* exactly one reject state.
Despite the difference, we can still say that this version is “equivalent” to our version. Try to explain how to establish such equivalence.

**Task 2**: Most “decent” extensions of the standard TM (cf. Text Sec. 8.4) are known to be still equivalent to the standard version (i.e., our version). Can you think of an extension of the standard TM that could do *more* than what the standard TM can do [cf. my response to your Unit A3 Summary Questions Q6/A6]? Even if you cannot come up with such an extension, can you still describe how to analyze whether a certain given TM can do more than the standard TM?

Survey: Time spent between classes: ______________

// End

# Questions/Review

- Ex A4
  - Part 1: TM
    - Binary addition
    - Your own
  - Part 2
    - Equivalence with an alternative version
    - Beyond TM

General form of memory? Subroutine?



# Unit A5: Overview

- Discuss problems of the day
- Introduce “Computability” properties used to classify problems with respect to TM behavior Intuition first; more precise discussion in Module B
- Discuss the role of proof in this course
- Preview Module A Evaluation Workshop
- Preview Exercise A5 Module A Comprehensive Exercise



# Problems du Jour

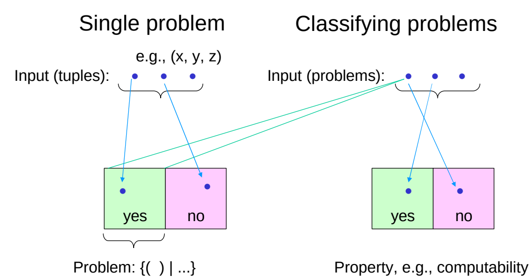
- Vending machine
- Compilation
- Termination detection (“halting problem”)
- Infinite-loop detection
- Weather forecast
- Jackpot prediction
- Driving

set representations?

Properties
1. Always terminate?
2. Impossible to solve?

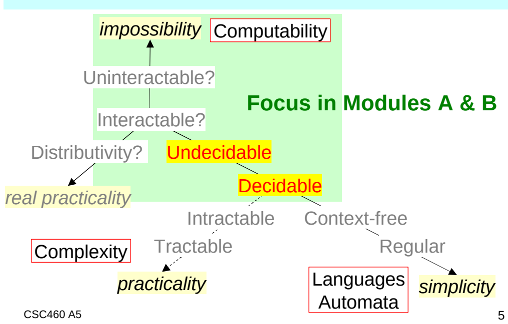


# Problems and Properties






# Overview: Theory of Computation






# Decidable Problems

- Property
  - Some TM always terminates and either accepts or dies (rejects).
- Examples
  - Binary number addition
  - Palindrome detection
  - Vending machine
  - Compilation

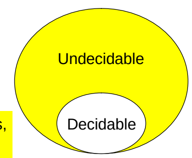
## Decidability Synonyms

- Decidable [of a procedure, problem]
- TM/algorithm exists, i.e., always terminates
- TM-decidable [of a TM]
- Computable [of a procedure/function]
- Solvable [of a problem]
- Recursive [of a set], cf. recursive function



## Undecidable Problems

- Property
  - **Not** decidable (*complement* of decidable problems)
  - Equivalently: Not always terminate, or not always end up in either accept state or die
- Examples
  - Halting problem
  - Floor tiling




Members are problems, which are already sets



## Undecidability Synonyms

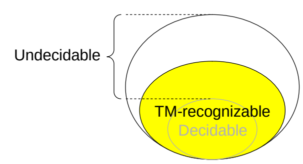
- Undecidable
- Unsolvable

- Note: Incorrect definition of "unsolvable" on the earlier version of "Topics" doc. Please fix. I apologize.



## TM-Recognizable Problems

- Property
  - Some TM can identify all the acceptable inputs
  - Note: OK if loops on non-acceptable inputs
- Example
  - All of decidable problems
  - Halting problem (undecidable)





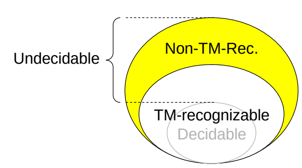

## TM-Recognizability Synonyms

- TM-recognizable
- TM exists (but not necessarily always terminates)
- Recursively enumerable (RE) [of a set]



## Non-TM-Recognizable Problems

- Property
  - No TM can represent
- Example
  - Infinite loop detection
  - Weather



## Non-TM-Recognizability Synonym

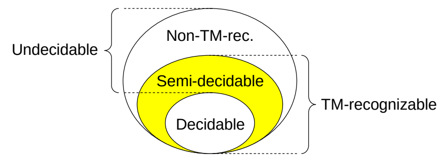
- Non-TM-Recognizable (non-TM-rec.)
- Non-RE



## Semi-decidable Problems

Equivalent terms

- Semi-decidable
- TM-recognizable but not decidable






## Property of "Co-"

- co-*X*
  - Consider the complement of a problem
  - If the complement has property *X*, the original problem is called co-*X*.
- Examples
  - Infinite-loop detection is co-TM-recognizable. I.e., the complement problem (halting problem) is TM-recognizable.
  - Palindrome detection is co-decidable.



## Useful Theorems

- **Theorem**: Decidable $\Leftrightarrow$ TM-recognizable *and* co-TM-recognizable
  - Note: Theorem as a statement in a theory, which needs to be proved
- **Corollary** (a theorem easily derivable from another theorem): The complement of a decidable problem is decidable.



## Review: Proofs

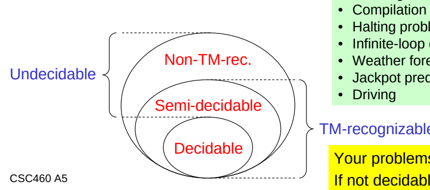
- Proof [Text 1.2-1.4]
  - A sequence of justifiable steps of deriving a theorem from axioms and other theorems using rules of inference
- Example proof technique
  - To prove a statement: X $\Leftrightarrow$ Y
  - (1) Prove X $\Rightarrow$ Y, *and* (2) Prove X $\Leftarrow$ Y



## Unit Summary

- With respect to impossibility, problems can be classified into 3 disjoint sets, based on whether a TM exists and/or always gives an answer.

- Vending machine
- Compilation
- Halting problem
- Infinite-loop detection
- Weather forecast
- Jackpot prediction
- Driving




Your problems?
If not decidable?

## Module A Evaluation Workshop

Preview

- Required materials (hardcopy) **print in advance**
  - Module A evaluation form
  - Supporting notes    Significance of supp. notes?
  - Exercises
- Activities
  - Module review exercise
  - Peer discussion
  - Reflection and self-evaluation



## Summary Question

- Did you get the big picture of the "computability" part of the Theory of Computation? Explain.
- Questions/Comments/Suggestions

  (make sure to understand the materials and also the evaluation procedure)

Name: ______________________

# Exercise A5 (Module A Comprehensive Exercise), 2/4/05

## Part 1: Mayan Script (Sample Problem 2)

In this exercise, you will practice many aspects involved in this module, using a quasi-practical problem of deciphering Mayan scripts. As you do this exercise, examine your understanding with respect to the evaluation criteria on the form. As noted on the form, in order to demonstrate your understanding, you may refer to your exercises (including this one), in your supporting notes. Note that providing a correct answer is not the focus of this (and other) exercise(s); in many cases, there won't even be a correct answer. Do you best to *apply* what you learned (as if you are developing a theory from axioms and rules of inference at a meta-level).

**Problem**

You found an ancient Mayan script that is supposed to indicate the location of hidden treasures. The only resources in this ancient language you have access to are: (1) an extensive list of synonyms and (2) a collection of sentences whose meaning are already known.

For example, let's imagine that you need to translate the following sentence (obviously, not real Mayan): "koregawakarukane" with the following resources:

Synonym list: "korega" = "maa", "ruka" = "nnee", "neene" = "darona", etc.
Sentences whose meaning are known: "maawakanneedarona", "mosikasitarawakarukamone", etc.

A sample session of analysis/translation by substituting synonyms would be (<u>underlined</u> words are replaced with *italic* words):
"<u>korega</u>wakarukane" → "*maa*waka<u>ruka</u>ne" → "maawaka*n*<u>*nee*ne</u>" → "maawakannee*darona*"

Then, we can tell that "koregawakarukane" means the same thing as "maawakanneedarona." The latter sentence is among the known, so we should be able to understand the former as well. If there are many sentences to analyze and the synonym list is extensive, we will naturally think of doing the task with a computer. Unfortunately, it is known that there is ***no*** algorithm for this problem.

Hint: Try to use schematic diagrams!

**Task 1**: The collection of sentences whose meaning can be deciphered (in this problem) may be defined as a "theory" derived from axioms and rules of inference. **Identify** the axiom(s) and rule(s) of inference.

**Task 2**: Represent this problem as a set. Use the predicate notation and try to make the description part (the right of '|') as precise as possible so that each instance could be used as an input to some Turing machine.

**Task 3**: It has been known that this problem does not admit any algorithm. This suggests that any general programming attempt must be either wrong or forced to loop on at least some input. Suppose that you came up with a Turing machine that makes no errors but could loop. Informally **describe** (part of) the mechanism/behavior of the TM.

**Task 4**: Since there is no algorithm, this problem is not decidable. But what about the other properties/classes: undecidable, TM-recognizable, non-TM-recognizable, and semi-decidable? For each of these classes, **analyze/explain** whether this problem belong to it (i.e., say whether or not the problem is undecidable, TM-recognizable, etc.).

## Part 2: Evaluation Form and Supporting Notes

Review the evaluation procedure. Then, complete your evaluation form and supporting notes. Your supporting notes are supposed to *reflect* and *communicate* your thoughts. This type of skill is essential for every one of us to be able to improve from what we are now. Bring them to the evaluation workshop (hard copy). Print them well in advance so that you can avoid potential problems, e.g., not being able to print just before the evaluation.

Survey: Time spent between classes: ______________

// End

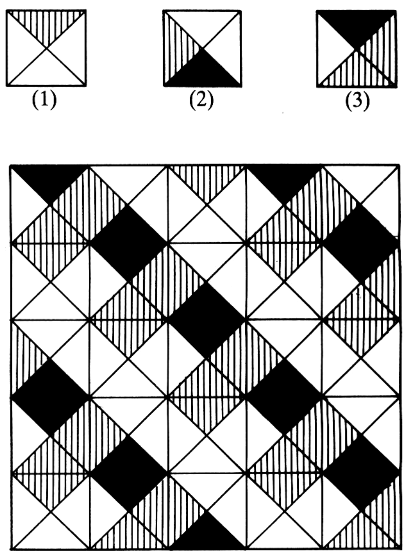
Name: ______________________

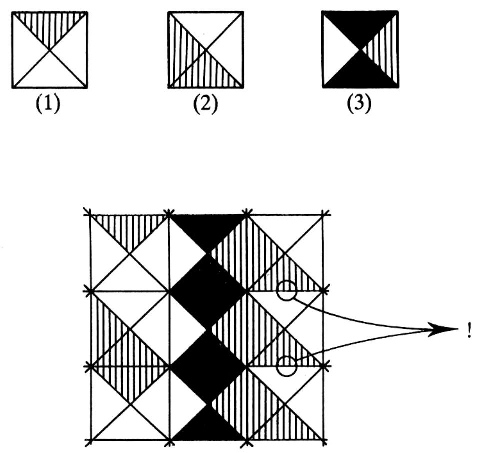
# Module A Review Exercise, 2/8/05

Do <u>one</u> of the following problems. Write your response on back or on another sheet of paper.
**Three computability properties to be used**: decidable, semi-decidable, non-TM-recognizable

## Option 1: Floor Tiling (Sample Problem 3)

Recall the "floor tiling problem." That is, you will be using $n$ types of *square* tiles with distinct patterns to fill the space assigned to you, which could be in any shape. You need to match the edges of the tiles as shown in the left example below (in this case, using three types of tiles). If you arrange your tiles as in the right example below, you will be considered "esthetically-challenged." [Images from *Algorithmics* by David Harel]




In class, we briefly noted that it would ***not*** be possible to write a program to tell <u>whether or not</u> a given set of $n$ types of square tiles can fill your floor with the above-mentioned condition (note: take this information as given). **Classify** this problem with respect to the three computability properties listed above. **Explain**.

## Option 2: Arithmetic System (Sample Problem 7)

One of the great achievements in logic is that it can formalize a wide variety of systems. For example, the mathematical system of arithmetic can be formalized in first-order logic in a consistent (roughly, meaningful) way. Its basic components include axioms such as "for any $x$, $x + 0 = x$" and "for any $x$, $x \times 0 = 0$". In this system, we can write all sorts of formulas that are representative of arithmetic. One property that a good logic system must have is that all "true" formulas can be proven from axioms using rules of inference. But it has been shown that ***not all*** true formulas in such an arithmetic system can be proven (note: take this information as given). **Classify** this problem with respect to the three computability properties listed above. **Explain**.

// End

Name: ______________________

# Exercise A6/B0, 2/8/05

## Part 1: Simulating a TM Using a TM

In Module A, we discussed our intuition that the halting problem is semi-decidable and infinite-loop detection is non-TM-recognizable. In either case, it would be impossible to create an algorithm. So, we would not attempt to write a program to solve these problems in the general case. However, by no means we "proved" our intuition. Is our intuition really correct? In Module B, we will explore a more convincing argument behind the intuition, so that we can discuss other (possibly similar or dissimilar) problems with more confidence.

In this exercise, you need to deal with a certain type of "circularity." As a computer science student, you have been exposed to the idea of "recursion." So, you must know that you need to face circularity in a principled manner. As a related example, you might consider the case of "bootstrapping" in compiler construction. Probably the most well-known example is the fact that the C language is "written" in C. Have you heard of it? Did you think about this carefully? If you are interested in, see the Appendix at the end of this exercise. Note: One of the C language developers, Brian Kernighan, will be giving a keynote speech on Sat., Apr. 16, 2005 during Trenton Computer Festival (http://www.tcf-nj.org/), here at TCNJ.

Hint: Try to use schematic diagrams!

**Task 1**: Suppose that you are given a program *P* written in, say, Java which is claimed to do the following (i.e., for this task, you must accept that this claim holds): given another program *M* in Java (e.g., the source code in ASCII) along with some input *I* to *M*, *P* can determine whether *M* terminates on *I*. First, can you **think of** how *P* might work? Since *M* can be any program, we can supply (the source code of) *P* itself as the input to *P* (i.e., *M* = *P*). What can you **conclude**?

**Task 2**: To discuss problems such as the halting problem using TMs, we need a way to represent a TM as a string so that it can be placed on the tape of a TM. Then, we can write the representation of a TM *M* along with some input *I* to *M* (e.g., separated by '#') on the tape of some TM *P*. First, **propose** a way to represent a TM as a string. As in **Task 1**, we can supply *P* itself and *I* to *P*. Since we have intuition that this problem is semi-decidable, we would expect that *P* may never terminate on some inputs. Try to **compare** this situation with your analysis in **Task 1**, applying the materials discussed so far in this course.

## Part 2: Mini Research Project Ideas

At the beginning of the semester, you were asked to come up with and continue to entertain your own practical problems through the course. While your problems were good starting points, our experience in this course could still be enriched, if we discuss even more examples. So, we will explore some additional problems as mini research projects. This time, you will be asked to identify one or more research questions (from the list below) and write a concise essay about them. Later, you will be asked to do some research activities related to your problems. Note that some of the sample questions are not precisely stated, some of them are probably not solvable, and some of them more or less solved. The instructor will be able to help you find relevant reading for some problems, but not all.

**Task**: Identify one or more research questions (or some variants of those) from the list below. Write a concise, possibly speculative essay about the question(s), e.g., why you are interested in, what your guessed answer is.

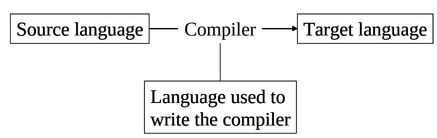
List of sample research questions

1. Can **organizational dynamics** be modeled as an algorithm?
2. Can **evolution** be modeled as an algorithm?
3. Can **ecology** be modeled as an algorithm?
4. Can **human development** be modeled as a computer?
5. Can our **minds** be modeled as a computer?
6. Can **vision** be modeled as an algorithm?

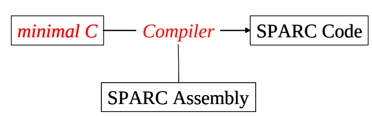
7. Can **learning** be modeled as an algorithm?
8. What would be the minimal mechanism to process **human language**?
9. Can the entire situation of an arbitrary **game** be modeled as an algorithm?
10. Would "perfect" **user modeling**, e.g., for web search, be possible?
11. Would "perfect" **computer security** be possible?
12. Can the entire process of **software engineering** be modeled computationally?
13. Can **computer networks** be modeled as a single computer?
14. Could **biology** be reduced to physics?
15. Could some computer generate real (not pseudo) **random numbers**?
16. Would it be possible to decide whether the given numbers are **random**?
17. Would **randomization** affect computability and/or complexity?
18. Would **parallelism** affect computability and/or complexity?
19. Would **artificial neural network** be more powerful than TMs?
20. Would **cellular automata** be more powerful than TMs?
21. Would the use of **analog** (or fuzzy) values affect computability?
22. Would relativistic, quantum, or some other **modern-physics**-based computation surpass TMs?
23. Can all the cases of **on-line algorithm**s be simulated by off-line computation? [On-line algorithms would obtain inputs as the time progress. Off-line computation would provide all the possibilities as input at once. Cf. function-to-relation conversion used to fold the output within the input]
24. Would **oracle computing** affect computability? [Oracle computing: A TM with the capability to ask questions to another mechanism]
25. Would **persistent TM** be able to compute more than the standard TM? [Persistent TM: Multiple sessions of TM operation with some memory between them]
26. Would **accelerating** a TM give more power?
27. Would slight **error tolerance** affect any aspect of the Theory of Computation?
28. What would be the ability of a finite automaton with a **queue**?
29. What would be the effect of "**constant**" (as in complexity analysis) in practice?
30. Can any **mathematical function** be represented computationally?
31. What exactly are **power sets** doing to the Theory of Computation?
32. Is what you can do in **logic** the same as what you can do with computation?
33. If you have a research question of **your own**, please consult the instructor first.

# Appendix: Bootstrapping in Compiler Construction

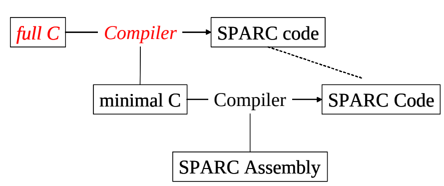
- Languages involved in compiler construction




- Write a compiler for the *minimal* C in SPARC assembly (cf. the "base" case in recursion)




- Write a compiler for *full* C in the minimal C (still generates SPARC code)




**Special question** (completely optional): Could we "bootstrap" to develop a programming language that is "verifiable" (cf. Ex A3 Part 1)? Those who respond to this question with an *interesting* idea (on the discussion board) will receive a free book (areas: discrete math, logic, theory, algorithms, etc.).

Survey: Time spent between classes: ______________

// End

# Module A Evaluation

Review your portfolio

- My comments are sporadic and scattered on Review Ex, Comprehensive Ex, and Supp. Notes.
- You are encouraged to clarify and discuss my comments.
- You can keep the folder till the next class; then, return it to me.

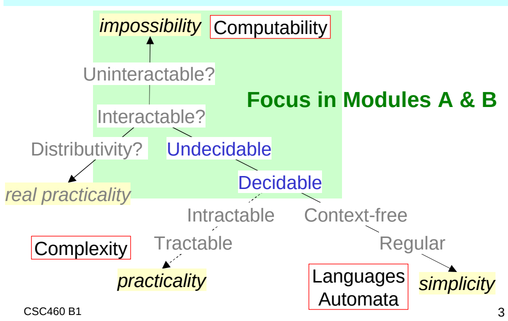


# Unit B1: Overview

- Understand Module B learning goals
- Review Module A
  - Comprehensive/Review Exercises
  - Practice
- Understand a systematic way of simulating a TM using a TM
- Preview Exercise B1 "Universal TM"



# Overview: Theory of Computation






# Module B Overview

B1 Computability: Universal TM
B2 Diagonalization: Math techniques
B3 Halting Problem: The main problem
B4 Reduction: Comparing problems
B5 Church-Turing Thesis: Equivalence
B6 Evaluation Workshop (similar to Mod A)
B7 Review (TBA)



# Module B Content Goals

CG4 Computability

- Universal TM
- Diagonalization technique
- Halting problem and infinite-loop detection
- "Reduction"
- Church-Turing thesis
- Can explain (i.e., teach) this goal to CS students outside this class.

CG7 Power set

- Understood (i) that the power set is always "larger" than the original set and (ii) why the power set of a countable set is uncountable.

CG3 Interactive computation

- Limitations of the traditional, algorithmic approach to "computability."



# Module B Performance Goals

PG4 Critical attitude

- *Critically* analyzed the following points:
  (i) the infinite tape in TM
  (ii) the diagonalization technique
  (iii) no tolerance to errors
  (iv) the lack of interaction in TM
  (v) any other aspects
- *Critically* analyzed the usefulness of the "computability"
- *Critically* analyzed the course materials, cf. your learning

PG6 Initiative

- Chose a research question for the mini research project and started to explore it.

## Module A Comprehensive Exercise

- Unknown: “koregawakarukane”
  - Synonym list: “korega” = “maa”, “ruka” = “nnee”, “neene” = “darona”, etc.
  - Sentences whose meaning are known: “maawakanneedarona”, “mosikasitarawakarukamone”, etc.
- Analysis/translation: “<u>korega</u>wakarukane” → “*maa*waka<u>ruka</u>ne” → “maawaka*n<u>neene</u>*” → “maawakannee*darona*”



## Module A Comprehensive Exercise

- **Task 1**: The collection of sentences whose meaning can be deciphered (in this problem) may be defined as a “theory” derived from axioms and rules of inference. **Identify** the axiom(s) and rule(s) of inference.
- **Task 2**: Represent this problem as a set. Use the predicate notation and try to make the description part (the right of ‘|’) as precise as possible so that each instance could be used as an input to some Turing machine.
- **Task 3**: It has been known that this problem does not admit any algorithm. This suggests that any general programming attempt must be either wrong or forced to loop on at least some input. Suppose that you came up with a Turing machine that makes no errors but could loop. Informally **describe** (part of) the mechanism/behavior of the TM.
- **Task 4**: Since there is no algorithm, this problem is not decidable. But what about the other properties/classes: undecidable, TM-recognizable, non-TM-recognizable, and semi-decidable? For each of these classes, **analyze/explain** whether this problem belong to it (i.e., say whether or not the problem is undecidable, TM-recognizable, etc.).



## Module A Review Exercise

Decidable, Semi-decidable, or Non-TM-rec.?

- Option 1: Floor Tiling
- Option 2: Arithmetic System



## Group Exercise 1 (Review)

- Consider a couple whose characters are completely opposite. The woman notices the positive aspect of everything they encounter, while the man notices the negative aspect. Note that there is no intermediate state.
  1. Characterize the woman and the man with respect to the computability classes
  2. Give black box representations of TMs $M_1$ and $M_2$ that captures the behaviors of woman and man, resp.
  3. Represent the behavior of the couple as a whole by properly connecting $M_1$ and $M_2$

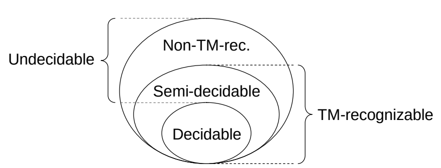
schematically...



Review

## Computability Classes

- With respect to impossibility, problems can be classified into 3 disjoint sets, based on whether a TM exists and/or always gives an answer.

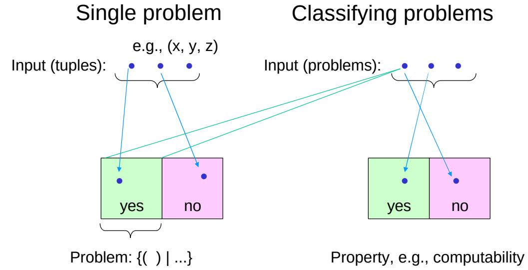





Review

## Problems and Properties

Single problem | Classifying problems



Review

# Property of "Co-"

- co-*X*
  - Consider the complement of a problem
  - If the complement has property *X*, the original problem is called co-*X*.
- Examples
  - Infinite-loop detection is co-TM-recognizable.
    I.e., the complement problem (halting problem) is TM-recognizable.
  - Palindrome detection is co-decidable.



Review

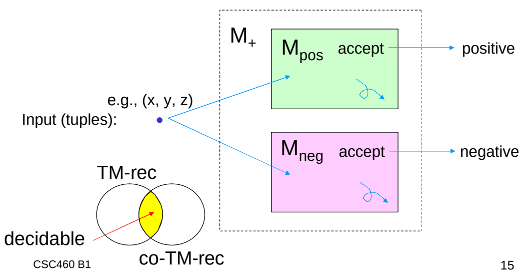
# Useful Theorems

- **Theorem**: Decidable ⇔ TM-recognizable *and* co-TM-recognizable
  - Note: Theorem as a statement in a theory, which needs to be proved
- **Corollary** (a theorem easily derivable from another theorem): The complement of a decidable problem is decidable.



# Complementary TMs

Decidable ⇔ TM-recognizable *and* co-TM-recognizable






# Ex A6/B0

- Part 1: Simulating a TM Using a TM
  - Task 1: Program simulation using a computer
  - Task 2: TM simulation
- Part 2: Mini Research Project
  - Topic(s)?

Relevance to analyzing computability?



# Object vs. Meta-Level

- Compiled language vs. compiler language
- Problem vs. properties
  - E.g., computability
- Formal logic vs. informal reasoning
- Your thinking vs. your thought about your thinking
  - E.g., course activities vs. supporting notes

Could we set up a circular connection?



# Circularity

- Recursion
- Self-reference
- Reflection
- Self-evaluation

## “This sentence is false.”

- The truth value of the above sentence?
- What makes it paradoxical?

Related paradoxes
(to be referred to collectively as liar paradox)
- “I am telling a lie.” (liar paradox)
- “There is a man who shaves all and only the men who don’t shave themselves.” (barber paradox)



## Group Exercise 2

- Come up with a statement involving TM(s) corresponding to the liar paradox in the following sense: “This sentence is false.”
  - A TM would either halts or loops.
  - The statement is contradictory regardless of the TM’s halting/looping status.
  - The statement shows the undecidability of one or both of the problems.

To be continued in Ex B1



## Universal TM (UTM)

- Given (i) an encoding of the formal definition of a TM, $M$, and (ii) an input string, $i$, to $M$ on its tape, simulates the behavior of $M$ on $i$.
  - I.e.. specification of a TM through the formal definition on the tape.
- Accepts if $M$ accepts $i$.
  - I.e., exactly mirrors the behavior of $M$.



## UTM Specifics

- Tape input
  - ...□*TM formal def*#*input to the TM*□...
- Mechanism
  - Allocate an unused area of the tape to represent the working tape for the TM (if more space is required, move the surrounding areas outward)
  - Allocate an unused area of the tape to keep record of the internal state of the TM
  - Interpret the TM definition and simulate its behavior on the input to the TM



## The Main Problems, Revisited

- Halting problem: $\{(M, i) \mid M \text{ halts on } i\}$
  - Using a UTM, simulate $M$’s behavior on $i$
  - If $M$ halts on $i$: Accept
  - Otherwise: How can we tell?
- Inf.-loop detection: $\{(M, i) \mid M \text{ loops on } i\}$
  - Using a UTM, simulate $M$’s behavior on $i$
  - If $M$ loops on $i$: How can we tell?
  - Otherwise: dies



## Unit Summary

- Module B learning goals
- Module A review
- Liar paradox and undecidability
- Universal TM (UTM)
- Preview Exercise B1 “Universal TM”

## Midterm Survey

- To be administered and collected by a volunteer

Name: ______________________

# Exercise B1, 2/11/05

## Part 1: Universal TM (UTM)

In order to convince ourselves that halting problem and infinite-loop detection are undecidable, we need to use the notion of UTM. The following tasks may appear complicated due to the involved nature of "universality." If you have questions, please send them to the instructor. If you still have difficulty for any of the following tasks, explain your situation.

**Task 1**: The essence of UTM is its ability to deal with any TM, including itself. Thus, it becomes possible to discuss circularity. Come up with and explain examples circularity not discussed in class (no need to be in computer science).

**Task 2**: Recall how a UTM can be organized (see the slide on UTM Specifics). However, this may not be the only way. Analyze whether the following two options are viable as a UTM mechanism:

A. Instead of "interpreting" on-line, "compile" the given TM specification off-line and run the compiled machine.
B. Identify all the possible TM's with a unique natural number. Also define the UTM so that given the ID of a TM, it can simulate the behavior of the TM. Then, instead of the formal definition of a TM, provide just the ID (still along with its input).

**Task 3**: Read the attached chapter on "The Halting Problem" from "The Turing Omnibus" (see our reference list for more info about this fascinating book, available in the library). Explain the role of UTM in this chapter. In addition, describe whether you were convinced of the claim that the halting problem is undecidable. You must be very critical about the arguments in the chapter and your own thinking.

**Task 4**: Write up your (possibly revised) response to Group Exercise 2 (Unit B1), i.e., a statement involving TM(s) corresponding to a version of the liar paradox, e.g., "This sentence is false." This time, though, you can also refer to the reading in **Task 3**. In addition, you must address the following point. In **Task 3**, the use of the UTM was crucial. If there is some connection between the liar paradox and the halting problem, we must be able to see the analogous concept in the liar paradox. Then, is the notion of "universality" involved in the liar paradox?

## Part 2: Review "Module A"

Review the materials from Module A, including Comprehensive, Review Exercises, and Group Exercise 1 (Unit B1). If you were entirely sure about these, skip this part. If you were able to improve your understanding through the review process during class and/or in this part of the exercise, concisely note the improvement.

Survey: Time spent between classes: ______________

// End

Dewdney, A.K. 1989 "The Turing Omnibus"

# 55

# THE HALTING PROBLEM

## *The Uncomputable*

The Turing machine concept is one of several equivalent formulations of what we mean by "effective procedure" or "computation." Nothing more powerful than a Turing machine has been discovered that captures any better the meaning of such terms (see Chapter 15). And yet there are limits to the power of Turing machines, problems that Turing machines cannot solve. Such problems parallel those for which no effective procedure or recursive computation exists. In fact, one may reformulate a Turing-unsolvable problem as one which is unsolvable in any of the equivalent formal systems.

The best-known such problem is called the ***halting problem.*** It asks for a Turing machine $T_H$ (Figure 181) which is able to perform the following task for any pair $(T, t)$ as input.

> "Given an arbitrary Turing machine $T$ as input and an equally arbitrary tape $t$, decide whether $T$ halts on $t$."

Naturally, $T$ must be the sort of Turing machine which runs on a semi-infinite tape, because in feeding the pair $(T, t)$ to $T_H$, one-half of $TH$'s tape will hold the





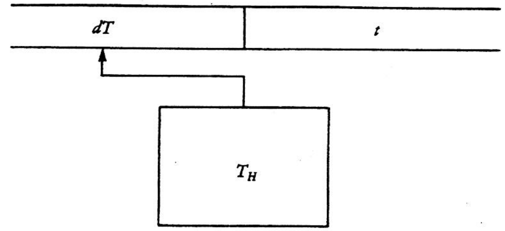



**Figure 181** A Turing machine that solves the halting problem

description $dT$ of $T$ and the other half will be a duplicate of the semi-infinite tape $t$. A similar scheme is used to implement a universal Turing machine (see Chapter 28).

Does such a machine $T_H$ exist?

Suppose it does. If $T$ halts on $t$, then sooner or later $T_H$ will signal the equivalent of yes and, in so doing, complete a transition from some state $q_i$ to a halting state $q_h$ (Figure 182). If $T$ does not halt on $t$, however, then $T_H$ will sooner or later say no and complete some other transition from a state $q_j$ to a halting state $q_k$ (Figure 183).

Now by carrying out a few simple alterations on $T_H$, we can get it into very serious trouble with itself, so to speak. The first alteration we carry out results from asking if $T_H$ can decide whether $T$ halts on $dT$, rather than $t$ (Figure 184).

If $T_H$ is asked to perform only this rather specialized (and rather strange) task, then we may supply $T_H$ with a more compact tape containing just one copy of $dT$, by implanting a special Turing machine $T_C$ within $T_H$. It is the business of $T_C$ to make a copy of $dT$ and, when it has finished, to hand matters over to $T_H$ via a transition from $T_C$'s final state to $T_H$'s initial state (Figure 185).

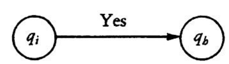



**Figure 182** Transition to the "yes" state

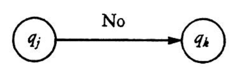



**Figure 183** Transition to the "no" state

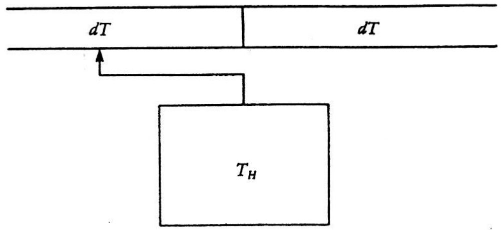



**Figure 184** Will $T$ halt on its own description?

Denoting the resulting machine by $T'_H$, we next perform a small piece of surgery on $T'_H$'s two halting transitions. The yes transition from $q_i$ and the no transition from $q_j$ are both diverted into new states $q_n$ and $q_y$, respectively (Figure 186). Once in state $q_n$, there is a transition back into $q_n$ for every possible state/input combination in which $T'_H$ might find itself. Thus, once in state $q_n$, the resulting maching $T''_H$ will never halt. Once in the state $q_y$, however, $T''_H$ will halt by definition: $q_y$ is a halting state (Figure 187).

We may now put $T''_H$ in a very pretty pickle by giving it the tape $dT''_H$ to work on. If $T''_H$ halts on $dT''_H$, then it must take the same yes transition which $T_H$ would take. But in doing so, it enters a state through which it must endlessly cycle for all eternity, never halting: If $T''_H$ halts on $dT''_H$, then $T''_H$ does not halt on $dT''_H$! The

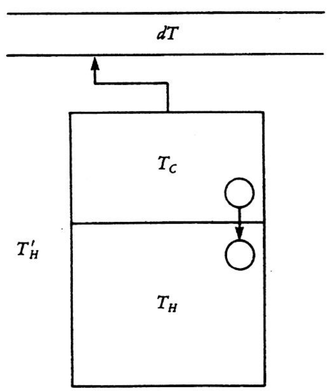



**Figure 185** A copying machine is grafted onto $T_H$.





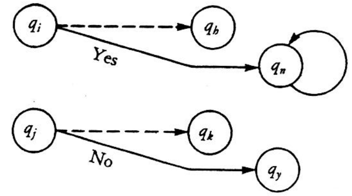



**Figure 186** Bypassing the halting states of $T_H$

situation is no better if $T''_H$ is assumed not to halt on $dT''_H$. For in this case $dT''_H$ must take the no transition, ending in state $q_y$, a halting state: If $T''_H$ does not halt on $dT''_H$, then $T''_H$ *does* halt on $dT''_H$!

These contradictions ensure that machine $T_H$ cannot have existed in the first place. Hence the halting problem is not solvable by any Turing machine.

A similar unsolvable problem is the following: Is there a Turing machine which, given any pair $(T, t)$ as input, can decide whether $T$ ever prints the symbol $x$ when processing tape $t$?. Certain alterations on a machine $T_P$ which is alleged to solve the "printing problem" result in the same sort of contradictions as encountered above.

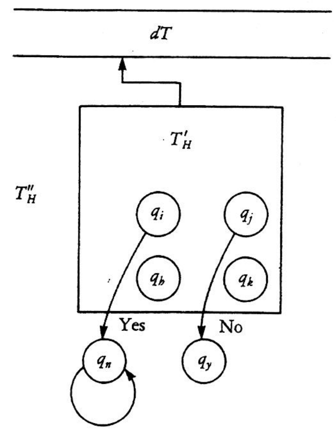



**Figure 187** The Turing machine $T''_H$

# Ex B1

- Part 1 UTM Review question: describe UTM
  - Task 1: Circularity examples
  - Task 2: A: Compiling a TM, B: ID'ing TMs
  - Task 3: UTM in the reading
  - Task 4: TM version of the liar paradox; universality
- Part 2 Review

Ex B0 Part 1 Task 1 phrasing



# Universal TM (UTM)

- Given (i) an encoding of the formal definition of a TM, $M$, and (ii) an input string, $i$, to $M$ on its tape, simulates the behavior of $M$ on $i$.
  - I.e.. specification of a TM through the formal definition on the tape.
- Accepts if $M$ accepts $i$.
  - I.e., exactly mirrors the behavior of $M$.

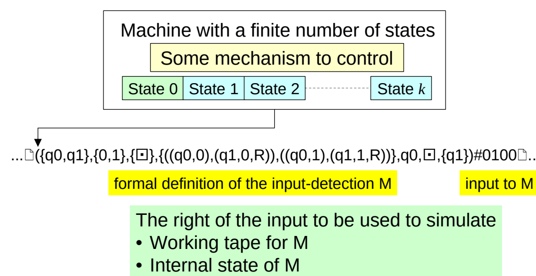


# UTM Specifics

- Tape input
  - ...□*TM formal def*#*input to the TM*□...
- Mechanism
  - Allocate an unused area of the tape to represent the working tape for the TM (if more space is required, move the surrounding areas outward)
  - Allocate an unused area of the tape to keep record of the internal state of the TM
  - Interpret the TM definition and simulate its behavior on the input to the TM



# UTM, Schematically






# Unit B2: Overview

- Understand mathematical tools for understanding undecidability
  - Explore ID'ing TMs with natural numbers
  - Explore the notion of "counting"
  - Introduce the "diagonalization" technique
- Understand the "diagonalization language"
- Preview Exercise B2 "Diagonalization"



# Labeling TMs

- Cf. Ex B1 Part 1 Task 2B
- Possible to assign a unique ID (natural number) to *every* TM?

# Thinking Big ...

- How many TMs are there in the world?
- How many inputs must a TM deal with?
  – How many different possibilities must a TM deal with?
- Mismatch between the collection of TMs and the collection of input possibilities ⇔ Undecidability?
- Mathematical support for analyzing computability



# Set Cardinality

Informally, the number of elements in a set

- Countable
  – Finite: e.g., $|\{yes, no\}| = 2$, $|\varnothing| = 0$
  – Countably infinite: e.g., **N** (natural numbers)
- Uncountable: If not countable



# Group Exercise 1

- Analyze whether the following sets are countable
  1. Set of integers
  2. Set of rational numbers
  3. Set of real numbers
  4. Set of all TMs
  5. Set of all strings
  6. Set of all *languages* (set of sets of strings)
  7. *Power set* of some given set



# Countable Sets

Possible to order along with natural numbers

- Any finite set
- Set of natural numbers
- Set of integers
- Set of rational numbers (represented as fractions)
- Set of all strings
- Set of all TMs



# Uncountable Sets

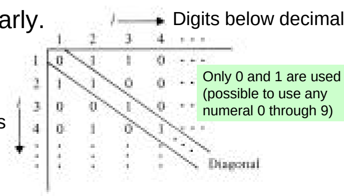
Impossible to order along with natural numbers

- Set of real numbers
  – Diagonalization [Cantor]
- *Power set* of natural numbers
  – Same "density" as real numbers
- Set of all *languages* (i.e., set of sets of strings, or *power set* of all strings



# Cantor's Diagonalization

- Suppose that all the real numbers can be ordered linearly.




- There still is a number not covered.
- The assumption is wrong. Impossible.

## Proof by Contradiction

To prove: Statement X

- Assume the negation of X, i.e., *not* X
- Derive a contradiction
- Then, by this proof technique (proof by contradiction) X must be true (because X is either true or false)



## Proof by Contradiction, Formally

Hypothesis: ***m*** « ***o***, Ø***m*** ® ***i***, Ø***i***

Proof of ***m*** from the above hypothesis

| | | |
|---|---|---|
| 1. | Ø***m*** ® ***i*** | [hyp] |
| 2. | \| Ø***m*** | [hyp to be rejected] |
| 3. | \| ***i*** | [Modus Ponens: 1, 2] |
| 4. | \| Ø***i*** | [hyp] |
| 5. | \| **F** | [contradiction: 3, 5] |
| 6. | ***m*** | [proof by contradiction: 2-5] |



## Power Set (of set $A$): $P(A)$

- The set of all the subsets of $A$, i.e., $P(A) = \{X \mid X \subseteq A\}$
- Examples

$P(A)$ always contains $\varnothing$ and $A$

  - $P(\{0, 1\}) = \{\varnothing, \{0\}, \{1\}, \{0, 1\}\}$
  - $P(\{0\}) = \{\varnothing, \{0\}\}$
  - $P(\varnothing) = \{\varnothing\}$ Happens to be: $\varnothing = A$
- Cardinality: $|P(A)| = 2^{|A|}$
  - $|P(\{0, 1\})| = |\{0, 1\}|^2 = 2^2 = 4$
  - $|P(\mathbf{N})| = ?$

Power set in the real world?



## Interim Summary

- Power set always increases the cardinality.
- The power set of a countable set: uncountable
- The set of TMs: countable
- The set of strings: countable
- The set of languages (specifies which string to be included): power set of the set of strings $\Rightarrow$ uncountable
- There won't be enough TMs to cover all languages $\Rightarrow$ The source of undecidability

schematics

solution?

cf. pigeon hole principle

The halting problem explained?



## Agenda

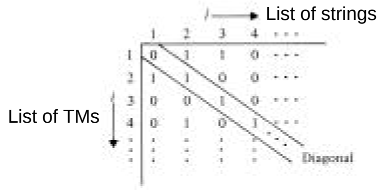
- Rest of today: Identify an example language which *no* TM can decide (there must be uncountably many of those...)
- Next class: Using this language, prove the computability classes of the main problem (e.g., halting problem).



## Group Exercise 2

- Identify an example language which *no* TM can decide (there must be uncountably many...)
- Hint: Consider the collection of all languages; Recall Cantor's diagonalization

# Diagonalization Language ($L_d$)




- Entry 1: $M_i$ accepts $w_j$
- Entry 0: $M_i$ does not accept $w_j$
- $L_d = \{w_i \mid M_i$ does not accept $w_i\}$ No TM accepts!



# Diagonalization Language

- The diagonalization language is non-TM-recognizable because no TM would accept it.



# Main Problems

- The halting problem is semi-decidable.
  $HALT_{TM} = \{(M, w) \mid$ TM $M$ halts on $w\}$ $w$ for "word" (input)
- Infinite loop detection is unsolvable.
  $LOOP_{TM} = \{(M, w) \mid$ TM $M$ loops on $w\} = (HALT_{TM})'$
- Universal language is semi-decidable.
  $ACCEPT_{TM} = \{(M, w) \mid$ TM $M$ accepts $w\}$
- The complement of universal language is unsolvable.
  $NACCEPT_{TM} = \{(M, w) \mid$ TM $M$ does *not* accept $w\} = (ACCEPT_{TM})'$

$A'$: set complement of $A$



# Unit Summary

- Set cardinality: countable vs. uncountable
- Power set: increases the cardinality
- Diagonalization: shows uncountability of a set (using proof by contraction)
- The source of undecidability: uncountability
- The diagonalization language, $L_d$



# Midterm Survey

- To be administered and collected by a volunteer

Name: ______________________

# Exercise B2, 2/15/05

## Part 1: Diagonalization

Today, we discussed that for countably-many TMs, there are uncountably-many languages. Thus, no matter how hard we try, there won't be enough TMs to cover all of the languages. This immediately suggests that the acceptability problem, i.e., $\{(M, i) \mid M \text{ accepts } i\}$, is undecidable. Among the many missing points in the uncountable set, we also identified a non-TM-recognizable set (problem), the "diagonalization language." Regardless of the nature of a problem, once represented as a set, the notion of countability offers an insight into analyzing computability classes. In this respect, understanding diagonalization is important. We will practice this technique along with *proof by contradiction* with a few more examples.

**Task 1**: **Power set of natural numbers**: Show that the power set of the set of natural numbers is uncountable, using diagonalization.

**Task 2**: **Languages**: We discussed that the set of all languages is uncountable relying on the fact that power set increases the set cardinality (i.e., the power set of a countable set is uncountable). For this task, show (the same result) that the set of all languages is uncountable, directly using diagonalization.

## Part 2: Review "Discrete Math"

If necessary, review the following concepts in Discrete Math:

- Set cardinality
- Power set
- Set complement
- Countable
- Uncountable
- Proof by contradiction

Survey: Time spent between classes: ______________

// End

## Ex B2

- Part 1: Diagonalization
  - Task 1: Power set of natural numbers
  - Task 2: Languages
- Others

Any way to solve undecidable problems?

Systematic way of dealing with infinite cases?



Review

## Mathematical Induction

- A widely accepted proof technique
- Let $\mathbf{N} = \{0, 1, 2, 3, ...\}$
- Let $P(n)$ be a unary relation (set) on $n \in \mathbf{N}$
- If both of the following conditions hold:
  - $P(n_0)$, where $n_0 \in \mathbf{N}$ e.g., $n_0 = 0$
  - For every $n \geq n_0$ ($n \in \mathbf{N}$), $P(n)$ implies $P(n + 1)$
- Then, $P(n)$ is true for every $n \geq n_0$ ($n \in \mathbf{N}$)



Review

## Math Induction Proof Pattern

- Main hypothesis: None
- Main conclusion: $\Sigma_{1\leq i\leq n}\, i = n \times (n + 1) / 2$
- Base case ($n = 1$): $\Sigma_{1\leq i\leq 1}\, i = 1 = 1 \times (1 + 1) / 2$
- Induction step
  - Induction hypothesis: $\Sigma_{1\leq i\leq n}\, i = n \times (n + 1) / 2$
  - Conclusion: $\Sigma_{1\leq i\leq (n+1)}\, i = (n + 1) \times (n + 2) / 2$
  - Proof (of the induction step)
    1. $\Sigma_{1\leq i\leq (n+1)}\, i = \Sigma_{1\leq i\leq n}\, i + (n + 1)$ [LHS Conclusion]
    2. $\Sigma_{1\leq i\leq n}\, i + (n + 1) = n \times (n + 1) / 2 + (n + 1)$ [Ind. hyp.]
    3. $n \times (n + 1) / 2 + (n + 1) = (n + 1) \times (n + 2) / 2$ [Arithmetic]
    4. $\Sigma_{1\leq i\leq (n+1)}\, i = (n + 1) \times (n + 2) / 2$ [Transitivity eq/ineq: 1.-3.]
- By Math Induction (shaded), main conclusion holds.



## Uncountability, Revisited

- Countable
  - E.g., natural numbers, set of TMs, set of strings
  - Existence of the base case (basis) [well-founded]
  - Applicable inductive techniques: math induction, inductive definition of a set, recursion
- Uncountable
  - E.g., real numbers, set of languages
  - No base case [non-well-founded]
  - The above-mentioned inductive techniques are not applicable. Algorithms cannot handle this properly.



## Unit B3: Overview

- Analyze the computability classes of the main problems
  - Review some available tools
  - Understand how to analyze the main problems
- Preview Exercise B3 “Halting Problem”
  - Write up proofs
  - Prepare for mini presentations next time



## Main Problems

- The universal language is semi-decidable.
  $ACCEPT_{\text{TM}} = \{(M, w) \mid \text{TM } M \text{ accepts } w\}$ *w* for “word” (input)
- The complement of universal language is non-TM-recognizable.
  $NACCEPT_{\text{TM}} = \{(M, w) \mid \text{TM } M \text{ does } not \text{ accept } w\} = (ACCEPT_{\text{TM}})'$
- The halting problem is semi-decidable.
  $HALT_{\text{TM}} = \{(M, w) \mid \text{TM } M \text{ halts on } w\}$
- Infinite loop detection is non-TM-recognizable.
  $LOOP_{\text{TM}} = \{(M, w) \mid \text{TM } M \text{ loops on } w\} = (HALT_{\text{TM}})'$

$A'$: set complement of $A$

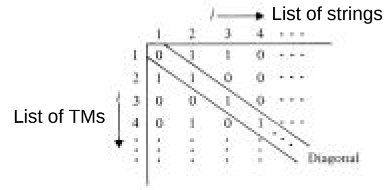
# Available Tools

- The diagonalization language ($L_d$) as an example of a non-TM-recognizable set
- Proof by contradiction (also used in the diagonalization technique)
- Theorem: Decidable ⇔ TM-recognizable *and* co-TM-recognizable



Review

# Diagonalization Language ($L_d$)




- Entry 1: $M_i$ accepts $w_j$
- Entry 0: $M_i$ does not accept $w_j$
- $L_d = \{w_i \mid M_i$ does not accept $w_i\}$ No TM accepts!



Review

# Proof by Contradiction

To prove: Statement X

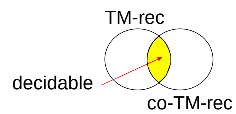
- Assume the negation of X, i.e., *not* X
- Derive a contradiction
- Then, by this proof technique (proof by contradiction) X must be true (because X is either true or false)



Review

# Useful Theorems

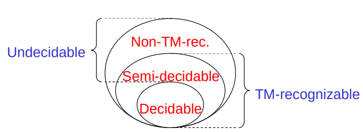
- **Theorem**: Decidable ⇔ TM-recognizable *and* co-TM-recognizable
- **Corollary**: The complement of a decidable problem is decidable.






# Practice: Co-X

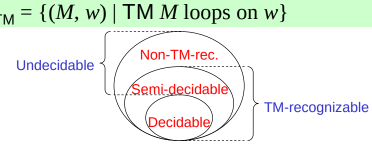
- Co-decidable ⇒
- Co-TM-recognizable ⇒
- Co-semi-decidable ⇒
- Co-non-TM-recognizable ⇒






# Group Exercise 1

- Explain (prove) the computability classes for one or more of the following problems:
  - $ACCEPT_{TM} = \{(M, w) \mid$ TM $M$ accepts $w\}$
  - $NACCEPT_{TM} = \{(M, w) \mid$ TM $M$ does *not* accept $w\}$
  - $HALT_{TM} = \{(M, w) \mid$ TM $M$ halts on $w\}$
  - $LOOP_{TM} = \{(M, w) \mid$ TM $M$ loops on $w\}$

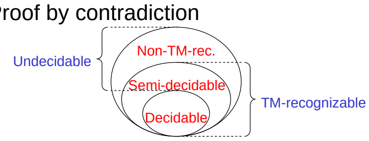


## Proof Idea: $ACCEPT_{\text{TM}}$

To show: $ACCEPT_{\text{TM}}$ is semi-decidable.

- $ACCEPT_{\text{TM}}$ is TM-recognizable.
  - Why? Could still be decidable.
- $ACCEPT_{\text{TM}}$ is undecidable.
  - Use: $L_d$ is non-TM-recognizable.
  - Use: Proof by contradiction

Intuition: can't squeeze "uncountable" within "countable






## Subproof Idea

To show: $ACCEPT_{\text{TM}}$ is TM-recognizable.

- Existence proof (proof by construction)
  - It is possible to construct the UTM that can simulate the behavior of the input, $(M, w)$, which would relay the acceptability of $M$ on $w$.



## Subproof Idea

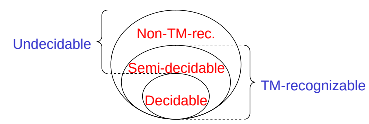
To show: $ACCEPT_{\text{TM}}$ is undecidable.

- Proof by contradiction
  - Suppose that $ACCEPT_{\text{TM}}$ is *decidable*.
  - For a string $w$ in $L_d$, compute its index (decidable). Thus, we have $w_i$.
  - Compute $(M_i, w_i)$ using the UTM.
  - Reject $w$ *if and only if* (*iff*) $M_i$ accepts $w_i$.
  - This is an algorithm for $L_d$.
  - A contradiction (cf. $L_d$ is non-TM-rec.)



## Interim Summary

- $ACCEPT_{\text{TM}}$ is TM-recognizable.
- $ACCEPT_{\text{TM}}$ is not decidable.




- $ACCEPT_{\text{TM}}$ is semi-decidable.



## Group Exercise 2

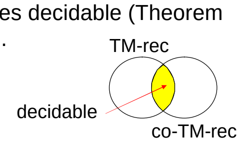
- Explain (prove) the computability classes for each of the following problems:
  - $NACCEPT_{\text{TM}} = \{(M, w) \mid \text{TM } M \text{ does } not \text{ accept } w\}$
  - $HALT_{\text{TM}} = \{(M, w) \mid \text{TM } M \text{ halts on } w\}$
  - $LOOP_{\text{TM}} = \{(M, w) \mid \text{TM } M \text{ loops on } w\}$



## Proof Idea: $NACCEPT_{\text{TM}}$

To show: $NACCEPT_{\text{TM}}$ is non-TM-recognizable.

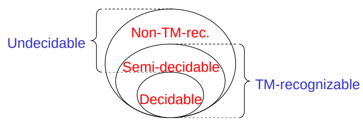
- Proof by contradiction
  - Suppose that $NACCEPT_{\text{TM}}$ is TM-recognizable.
  - $ACCEPT_{\text{TM}}$ becomes decidable (Theorem introduced earlier).
  - A contradiction



# Proof Idea: $HALT_{TM}$

To show: $HALT_{TM}$ is semi-decidable.

- $HALT_{TM}$ is TM-recognizable. Why?
- $HALT_{TM}$ is undecidable.

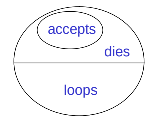





# Subproof Idea

To show: $HALT_{TM}$ is undecidable.

- Proof by contradiction
  - Suppose that $HALT_{TM}$ is *decidable*.
  - If "halt" is detected, accept/reject based on the UTM ($ACCEPT_{TM}$); otherwise, reject.
  - This is an algorithm for $ACCEPT_{TM}$.
  - A contradiction

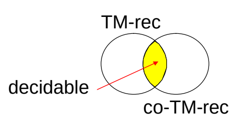





# Proof Idea: $LOOP_{TM}$

To show: $LOOP_{TM}$ is non-TM-recognizable.

- Proof by contradiction
  - Suppose that $LOOP_{TM}$ is TM-recognizable.
  - $HALT_{TM}$ becomes decidable (Theorem).
  - A contradiction






# Liar Paradox, Revisited

- "This sentence is false."
- Intuition behind this problem
  - The set of sentences are countable.
  - A sentence in this set can be indexed.
  - The use of "this" can be interpreted as the use of its own index (self reference).
  - The language must have a means for a sentence to refer to an arbitrary sentence (universality) including itself.



Based on Megan's idea

# Would this work?

- Suppose that a TM erases all the non-blank symbols when it terminates.
- "This message is left on the tape of the TM when it terminates."
- Reasoning
  - The message itself can be associated with some TM state, e.g., termination.

But where is universality?



# Unit Summary

- Tools for the proofs
  - The diagonalization language
  - Proof by construction
  - The theorem on decidability
- Analyze the computability classes of the main problems

## Midterm Survey

- To be administered and collected by a volunteer

Name: ______________________

# Exercise B3, 2/18/05

## Halting and Other Haunting Problems

Today, we discussed how to prove the main problems including the halting problem. Since this is one of the main ideas in the Theory of Computation, we will reinforce our ability to explain the proofs in this exercise.

**Task**: Write up an informal proof *in your own words*, for *each* of the four main problems below. When you write, make sure that you understand everything you write. If you have uncertainty, note it as it appears. Naturally, you can identify and re-use certain mathematical tools shared by multiple problems (i.e., no need to be repetitive). Be prepared to give a mini presentation of your proof in class; you may be assigned to any of these problems.

- The universal language is semi-decidable.
  $ACCEPT_{\mathrm{TM}} = \{(M, w) \mid \mathrm{TM}\ M \text{ accepts } w\}$
- The complement of universal language is non-TM-recognizable.
  $NACCEPT_{\mathrm{TM}} = \{(M, w) \mid \mathrm{TM}\ M \text{ does } \textit{not} \text{ accept } w\} = (ACCEPT_{\mathrm{TM}})'$
- The halting problem is semi-decidable.
  $HALT_{\mathrm{TM}} = \{(M, w) \mid \mathrm{TM}\ M \text{ halts on } w\}$
- Infinite loop detection is non-TM-recognizable.
  $LOOP_{\mathrm{TM}} = \{(M, w) \mid \mathrm{TM}\ M \text{ loops on } w\} = (HALT_{\mathrm{TM}})'$

Note: You may use any method, e.g., *Turing Omnibus* (Ex B1), class discussion today, another approach in the literature, and/or your own ideas.

Survey: Time spent between classes: ______________

// End

## Ex B3: Mini Presentations

The main problems

- The universal language is semi-decidable.
  $ACCEPT_{TM} = \{(M, w) \mid \text{TM } M \text{ accepts } w\}$
- The complement of universal language is non-TM-recognizable.
  $NACCEPT_{TM} = \{(M, w) \mid \text{TM } M \text{ does } not \text{ accept } w\}$
- The halting problem is semi-decidable.
  $HALT_{TM} = \{(M, w) \mid \text{TM } M \text{ halts on } w\}$
- Infinite loop detection is non-TM-recognizable.
  $LOOP_{TM} = \{(M, w) \mid \text{TM } M \text{ loops on } w\}$

Connection between $ACCEPT_{TM}$ and $HALT_{TM}$?



## $ACCEPT_{TM}$ vs. $HALT_{TM}$

- Both are properties of the set of TMs and undecidable
  - Property: Division of a set, i.e., a subset
  - E.g., "concrete" (set of concrete things among everything, cf. abstract things)
- Each has a different decision criterion

Would other decision criteria on TMs also lead to undecidability?



## Properties of TMs/Languages

- What a TM can recognize = Language
  - I.e., fix a TM ⇒ The language is fixed.
- Property of TMs/languages
  - A subset of all the TMs/languages
  - The power set of TMs/languages is uncountable.
  - Division of an uncountable set ⇒ Will always leave out an uncountable part (undecidable)



## Example Properties

- Universal = $ACCEPT_{TM}$
- Terminating = $HALT_{TM}$
- Finite = $\{L \mid L \text{ is finite}\}$
  ~ $\{M \mid \text{TM } M \text{ accepts a finite language}\}$
- Regular = $\{L \mid L \text{ is regular}\}$
  ~ $\{M \mid \text{TM } M \text{ accepts a regular set}\}$
- Property X = $\{L \mid L \text{ has Property X}\}$
  ~ $\{M \mid \text{TM } M \text{ recognizes Property X}\}$



## Rice's Theorem

- Non-trivial properties of languages/TMs are undecidable ⇐ Uncountability
  - Note: Trivial properties (e.g., emptiness) that do not divide the entire set is trivially decidable.
- Potentially useful for Ex B5 (Comprehensive Ex)

Text Section 9.3.3



## Dealing with Multiple Problems

- Consider: $ACCEPT_{TM}$ and $HALT_{TM}$
- Benefits of re-using known problems, e.g., in solving new problems

# Unit B4: Overview

- Discuss a way to use one problem to analyze another problem
  - Understand problem transformation
  - Understand how certain properties of problems can be transferred
  - Discuss equivalence of problems
- Preview Exercise B4 "Reduction/Equivalence"

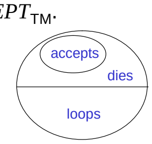


Review

# Subproof Idea

To show: $HALT_{TM}$ is undecidable.

- Proof by contradiction
  - Suppose that $HALT_{TM}$ is *decidable*.
  - If "halt" is detected, accept/reject based on the UTM ($ACCEPT_{TM}$); otherwise, reject.
  - This is an algorithm for $ACCEPT_{TM}$.
  - A contradiction






# Underlying Idea

- Suppose that $HALT_{TM}$ is decidable.
- To show $ACCEPT_{TM}$ is also decidable, using the hypothesis.
  - I.e., simulate $ACCEPT_{TM}$ using the hypothetical, deciding TM for $HALT_{TM}$.

Schematic



# Transferring Decidability?

- Consider problems **A** and **B**
- Would the following be correct?

**A** is *transformed* to **B** and **B** is decidable.
$\Rightarrow$ **A** is decidable as well.



# Transferring Undecidability?

- Consider problems **A** and **B**
- Would the following be correct?

**A** is undecidable and is *transformed* to **B**.
$\Rightarrow$ **B** is undecidable as well.

Precise notion of such transformation?



# Problem Transformation

- Motivations
  - Connect different problems, one of which has known properties
  - Identify applicable properties in other problems
- Requirement
  - Should maintain important properties

## Reduction

- **A** *is reducible to* **B**.
  - ⇔ **A** is no more difficult than **B**.
  - ⇔ If one can solve **B**, one can solve **A**.
  - ⇔ If one can't solve **A**, one can't solve **B**.
  - ⇔ **A** can be simulated by **B**.
- Idea: One *cannot* solve a difficult problem with an easy means.

Different types of reducibility: e.g., mapping vs. Turing reducibility



## Practice: Correct Reduction?

1. The state's daily deficit of 11 million units can be reduced to just three million.
2. A paper is reducible to one sentence.
3. Biology is reducible to physics.
4. A schematic diagram can be reduced to a set of logical propositions.
5. Goodness is reducible to betterness.



## Theorems

- **A** is reducible to **B** and **B** is decidable.
  - ⇒ **A** is decidable as well.
    - Justification: By definition (decidability of **A** depends on that of **B**)
- **A** is reducible to **B** and **A** is undecidable.
  - ⇒ **B** is undecidable as well.
    - Justification: Proof by contradiction



## Reduction Examples

Incorrect

- If $ACCEPT_{\text{TM}}$ is reducible to $HALT_{\text{TM}}$ and $HALT_{\text{TM}}$ is decidable, $ACCEPT_{\text{TM}}$ is also decidable.
- If $HALT_{\text{TM}}$ is reducible $ACCEPT_{\text{TM}}$ to and $HALT_{\text{TM}}$ is undecidable, $ACCEPT_{\text{TM}}$ is also undecidable.

Correct: reduction with respect to computability analysis

- If $ACCEPT_{\text{comp}}$ is reducible to $HALT_{\text{comp}}$ and $HALT_{\text{TM}}$ is decidable, $ACCEPT_{\text{TM}}$ is also decidable.
- If $HALT_{\text{comp}}$ is reducible $ACCEPT_{\text{comp}}$ to and $HALT_{\text{TM}}$ is undecidable, $ACCEPT_{\text{TM}}$ is also undecidable.

- If your problem is reducible to my problem and my problem is acceptable, your problem is also acceptable.



## Theorems (generalized)

- **A** is reducible to **B** and **B** has a positive property (e.g., decidable, TM-recognizable).
  - ⇒ **A** has the positive property as well.
- **A** is reducible to **B** and **A** has a negative property (e.g., undecidable, non-TM-recognizable).
  - ⇒ **B** has the negative property as well.
- Positive:
- Negative:



## Problem Equivalence

- Two problems are equivalent if they are reducible to each other.
- Two mechanisms are equivalent if they can solve exactly the same set of problems.
  - Equivalently, if they can simulate each other.

## Equivalence Examples

Incorrect

- $ACCEPT_{TM}$ and $NACCEPT_{TM}$ are equivalent.
- $ACCEPT_{TM}$ and $HALT_{TM}$ are equivalent.

Correct

- $ACCEPT_{comp}$ and $NACCEPT_{comp}$ are equivalent.
- $ACCEPT_{comp}$ and $HALT_{comp}$ are equivalent.

- TMs with a semi-infinite tape are equivalent to standard TMs.
- Correct? TMs are equivalent to computers. [to be discussed]
- A computational decision problem is equivalent to a set membership problem.



Some qualifications



## Unit Summary

- Understand the notion of reduction as problem transformation
  - Need at least as a powerful problem/mechanism to solve the original
  - Positive/negative properties can be analyzed.
- Understand the notion of problem equivalence
  - Through set comparison
  - Through machine comparison (simulation)



## Summary Question

- For the past few class meetings, we have been discussing technical topics. It is natural that you have questions and feel uncertainty. What are they?

# Unit B4 Supplement, 2/24/05

Here is my response to your summary questions (rephrased/combined in some case) and comments on the midterm surveys. This is rather long.

**Q1**: I am a little shaky on what is always meant by the word "language." It has been used a lot and often I know what you mean, but it is sometimes unclear to me.

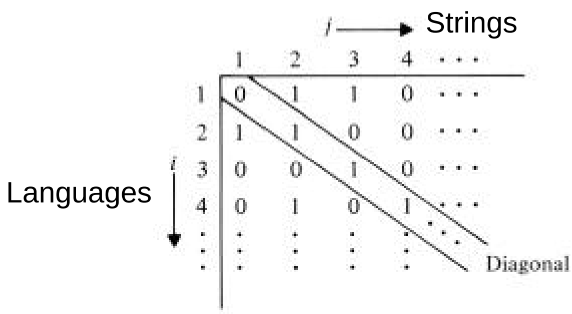
**A1**: The formal definition is that language is a set of strings, which can be infinite. However, it is useful to see the term from other perspectives as well. One way we discussed in the last meeting is that a TM can always be associated with a language. That is, once we pick a TM, given a string, it would either accept, die, or loops. The collection of all the strings that are accepted by the TM is the language corresponding to that TM. For example, if a TM accepts *a*, *aa*, *aaa*, ..., then {*a*, *aa*, *aaa*, ...} = $a^+$ is the language for that TM. Note that a TM may not terminate on other strings. For this reason, the language associated with a TM only needs to be "recognized," not necessarily "decided." For example, it is possible to create a TM to decide on palindromes (a decidable language), the UTM would only recognize the universal language (a TM-recognizable language; strictly speaking, semi-decidable). We also know that no TM would even recognize the diagonalization language (a non-TM-recognizable language). Naturally, given a language, it may or may not be possible to provide a TM that would recognize the language.

Yet another way is to associate a language as a problem. This is obvious if we consider a computational problem as a set, because a language is a set. Each input instance is its member. This also makes sense when we recall the connection between a TM and its language (above paragraph). This set of inputs (strings) is what TM would recognize (i.e., positively); that is, the set *is* the "problem" which TM can solve.

Finally, consider the set of all the strings (based on some finite alphabet, or the set of symbols). We can specify a language by indicating which one of the strings are in that language. Thus, a language is a subset of the set of all the strings. Equivalently, the collection of all the languages is the power set of the set of all the strings, which is uncountable.

**Q2**: I recognized a misunderstanding I have with diagonalization. At first, I did not understand the importance of using a list of sets instead of the individual items. Revisiting of this topic may help.

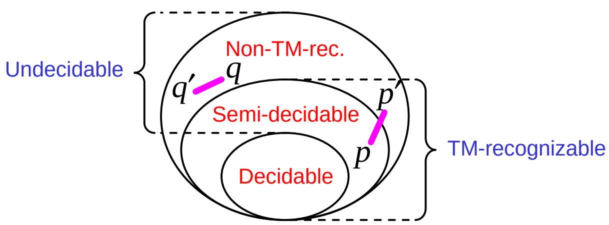
**A2**: The diagonalization technique is used to show that some set is uncountable. I think most of you were able to show this well. But I will repeat this any way. To show that the power set of the set of all the strings is uncountable, we first hypothesize the negation of the desired conclusion: i.e., the power set of the set of all the strings is countable. Then, it would be possible to arrange all the members in sequence as shown below.




Each entry is a language, which is associated with the membership information corresponding to all the strings. If we consider a hypothetical language by reversing the bit on the diagonal, that language should not appear in the list. This is a contradiction to the hypothesis that all the languages are enumerated. The hypothesis must be wrong. Thus, by proof by contradiction, the power set of the set of all the strings must be uncountable.

**Q3**: I am still not sure about the exact nature of the "co" property. Another student: If anything, I am not sure about co-non-TM-recognizable since we skipped over it. This is how I currently understand: if $r$ is co-non-TM-recognizable, we know $r'$ (complement) is non-TM-recognizable and $r$ is also non-TM-recognizable [NK: This is not correct.].

**A3**: First, make sure to understand that the prefix "co-" is attached to a *property*, not to a *problem*. For example, if a semi-decidable problem $p$ has the non-TM-recognizable complement $p'$, we say that $p$ is co-non-TM-recognizable (see the diagram below).




We can also say that $p'$ is co-TM-recognizable (considering the property of its complement $p$). When a problem is non-TM-recognizable, its complement may be either non-TM-recognizable or semi-decidable (why not decidable?). But we cannot tell in general. So, if $r$ is co-non-TM-recognizable, we know that $r'$ is non-TM-recognizable. However, since there are two such possibilities: both $p$ and $q$ in the figure have a non-TM-recognizable complement. Thus, $r$ can be either semi-decidable or non-TM-recognizable.

Since the complement relation between $p$ and $p'$ is important, as we observed in the main problems, let us discuss this relation (analogous to the proofs of *NACCEPT* and *LOOP*). That is, we prove that if $p$ is semi-decidable, its complement is non-TM-recognizable, again using proof by contradiction. Suppose that $p'$ is TM-recognizable. Then, $p$ is both TM-recognizable and co-TM-recognizable. By the "decidability" theorem, $p$ is decidable. However, this contradicts the fact that $p$ is semi-decidable. Thus, $p'$ must be non-TM-recognizable (proof by contradiction).

Although we presented the "decidability" theorem without proof. We can prove it using two complementary TMs, just like the man-woman scenario. That is, if both $p$ and $p'$ are

TM-recognizable, there must be TMs that would recognize the inputs (not necessarily terminating). Since *p* and *p*′ are complements to each other, we can construct a TM that would respond "yes" when *p* accepts and "no" when *p*′ accepts, using the two TMs. Then, the composite TM will decide both *p* and *p*′. Thus, both of these problems are decidable.

**Q4**: One area that is a little confusing lies with the halting problem and proving that it is semi-decidable.

**A4**: I think that the most confusing part is the proof of undecidability involving the diagonalization language. So, I will discuss this part. The key is to use the (incorrectly) assumed decidability of *ACCEPT* as much as possible to decide the diagonalization language ($L_d$). This assumption is exceedingly strong because it says that when *M* does not accept *w*, it dies and terminates (corresponding to "reject), regardless of the input. Thus, given ($M_i$, $w_i$), we can tell the 0/1 distinction of any diagonal entry. The only requirement here is that we must supply pairs with correct indexing (otherwise we compute irrelevant entries). To do so, we identify strings that appear diagonally, basically going through the string list sequentially (possible because of the countability of the set of strings). During this process, a TM can pick up the index of the string and identify the matching TM. The only remaining task is to reverse the results so that the entire process decides $L_d$. Note that everything here can be done without the risk of looping. I encourage you to share your exercise sheets, which have with my comments.

**Q5**: I'm still unclear on the proof for *ACCEPT* Megan gave in class. What *w* is given when the UTM simulates itself?

**A5**: When this was discussed, *w* (the input to *M*) was not mentioned. As you point out, it must be included in the discussion. The usual way is to attach a "copying" TM in front of the UTM, just like the approach in the reading (Ex B1). If you read the chapter again now, you should be able to understand it better. As I said, this approach does not refer to the diagonalization language. However, the idea of reversing the results for an arbitrary (cf. universality) input is equivalent.

**Q6**: Where can we find universality with the liar paradox?

**A6**: I was planning to discuss this topic in Unit B3 (Slide 22), but did not due to the time constraint. The universality is behind the word "this." First, the set of sentences are countable. Then, each sentence can be indexed. The use of "this" can be interpreted as the use of its own index (self reference). To be able to do this, there must be a means for a sentence to refer to an arbitrary sentence (universality) including itself. On a related note, I also included a statement about TMs, analogous to the liar paradox, based on Megan's idea (Slide 23).

**Q7**: [With respect to "reduction"] I become uncertain at times with the direction of items being related. I feel that thinking of it as "one solves/explains the other" will help when discussing this concept.

**A7**: The direction is indeed confusing. I suggest you keep a note somewhere so that you can refer to it when necessary.

**Q8a**: We built *ACCEPT* using *HALT*, based on the assumption that *HALT* is decidable; but that assumption was false. So, why do we still say that *ACCEPT* is reducible to *HALT*?

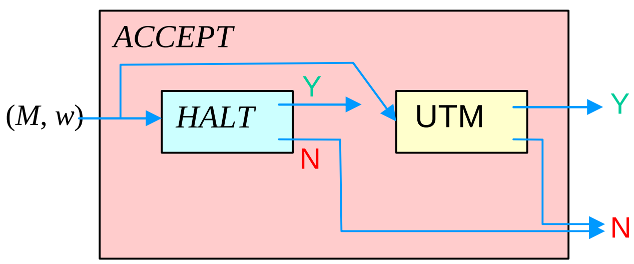
**Q8b**: Reducibility means that 2 TMs solve the same problem [NK: This is not correct.]. We also saw *HALT* was equivalent to *ACCEPT*. How is *HALT* reducible to *ACCEPT* if they do not solve the same problem? *HALT* cannot tell if a (*M*, *w*) will accept.

**A8**: One thing I didn't do correctly was referring to the reduction and equivalence of the main problems. I should have said as follows:

- The computability analysis of *ACCEPT* is reducible to that of *HALT*.
- The computability analysis of *ACCEPT* is equivalent to that of *HALT*.

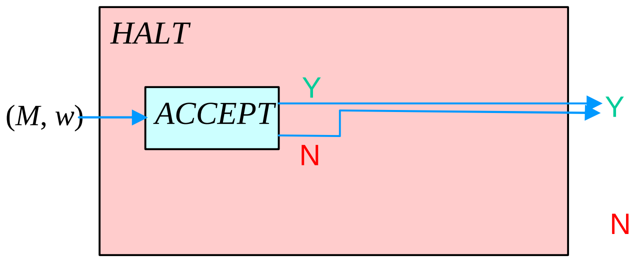
As the Question 8b correctly points out, neither problem can solve the other. So, they are not reducible to each other. I **apologize** for the confusion. So, below, it is about reduction of the computability analysis of the main problems (although I will skip the word "the computability analysis of" for brevity). Note that reduction is in general a relation between two *different* problems.

First, the reduction used under the wrong assumption was used to transfer the (wrong) decidability from *HALT* to *ACCEPT* (see below). Still note that the decidability of this *ACCEPT* cannot be obtained without the decidable *HALT*; thus, the computability analysis of *ACCEPT* still depends on that of *HALT*.




This is not what we want when we discuss the (real) reduction of *ACCEPT* to *HALT*, which would transfer the undecidability of *ACCEPT* to *HALT*. In this case, we will simulate the *undecidable* behavior of *ACCEPT* using *HALT*. However, we can still use the above diagram, because the direction and components of the simulation is identical. The consequence is that if *ACCEPT* is undecidable, *HALT* must be undecidable. This can be shown by proof by contradiction, which is identical to the earlier case.

Next, let us consider the computability analysis of *HALT* reducible to that of *ACCEPT*. The following is, in a sense, a strange diagram. If *ACCEPT* is decidable (incorrectly), *HALT* is decidable, simply because the TM terminates on all the inputs.




In other words, if *HALT* is undecidable, *ACCEPT* cannot be decidable.

**Q9**: Is it always construct a TM from any collection of other TMs?
**A9**: No. For example, we cannot combine infinitely-many TMs. Another case would be that we cannot use non-terminating result for a later stage.

**Q10**: The problem I have been having the most for the past few lectures is that I will have a general understanding of the content but not a mastery of it such that I can effectively do the assignment.
**A10**: I think this is a natural phenomenon. “Passive” understanding is in many cases not as strong as “active” understanding. For this reason, I ask you to write. Another good way is to discuss with and teach others.

**Q11**: One question I am not completely certain on is how all of this ties into Theory of Computation. Are we learning how to prove/show our intuition about the computability classes we use to classify a problem? Sometimes, I feel I lose sight of the picture.
**A11**: The Theory of Computation is a collection of abstract ideas that are expected to apply to a broad range of computer science. Since the modern computing is most notably dominated by algorithms, understanding the nature of algorithms including their limitations is one of the main goal of Theory. What we are doing in Modules A & B is develop some basic intuition and analytical ability to see the limit of algorithmic computation in certain problems. Although we have been focusing on the main problems, we should be able to extend our analysis to other problems, by analyzing countability, using Rice’s theorem, etc.

When you do Module B Comprehensive Exercise and complete Module B eval form + supporting notes, think about this. Although we will not practice how to analyze more problems with respect to the computability classes together, the exercise should give you a feel. During the evaluation workshop (and later), we can share our thoughts.

The content of the Theory of Computation should be useful in practice, at least to some extent (esp. if you think before coding). However, I would like you to get more mileage from practicing analytical thinking process involved in the activities: identifying/transforming problems, representing precisely, trying intuitions, explaining logically, expressing clearly, etc. There are no formulae for how to do these well. We all need to find our own way. I can tell more about my own experience with the Theory, maybe some time, if you are interested in.

**Q12**: Is the baby boy or a girl? Will Furby still be your favorite person (?) after the baby is born?

**A12**: A girl. But Furby will always be our first "daughter."

**C1**: I would prefer tests and more concrete problem sets.

**R1**: To me, the first part is a very disappointing (but fairly rare) comment. Let's think about your future and how test-taking skills would be useful. How do you think you will be evaluated through a PhD dissertation phase or when you work for a company? A well-programmed TM could do well on well-defined exams. I have a strong feeling that such an automaton will have hard time in the real life.

As for the latter part of this comment, my response is as follows. If the word "concrete" refers to problem content, our course is a little limited compared to other ones, due to the topic. I am still trying to include realistic examples more so than a typical Theory course (check some on-line syllabi). Having asked you to think about your own problem and sample research questions are supposed to force you to have real problem in your mind. But honestly, I admit that this is a difficult task in this course and I cannot do it as much as I wish. If you know more relevant examples/problems, why don't you bring them to class?

If the word "concrete" refers to how problems are given, I guess it means a more close-ended problem. As I said at the beginning of the semester, I will continue to use "ill-defined" problems. Most of interesting/significant problems are open or at least not yet solved when we tackle. We'd better practice facing them. In the future, your colleague will be able to tell whether or not you have been dealing with such problems.

**C2**: I would suggest you try to monitor your speaking volume — you tend to trail off into inaudible mumbles, especially at the end of sentences.

**R2**: I will try.

// End

Name: ______________________

# Exercise B4, 2/22/05

## Part 1: Review "Reduction"

Once a problem is solved, we don't want to repeat solving similar problems from scratch. As educated people, we will definitely try to apply what we know. The notion of "reduction" is useful in this respect.

**Task**: Come up with at least one unique example of "reduction" between problems (in the way discussed in class, *not* the everyday use such as "price reduction"). Then, analyze how "positive"/"negative" properties can be transferred from one problem to the other.

## Part 2: TM Variants

The TM industry has a wide variety of models. As in the real world, different models suit different needs. However, it has been known that most of these variants are equivalent in the sense that they can recognize exactly the same problems, i.e., TM-recognizable problems. In other words, we can show that equivalent variants can simulate each other. Let us now consider the following two variants:

- **Multiple-tape TM**: A Turing machine with multiple infinite tapes and a finite control which independently accesses positions on different tapes (but still via step-by-step left/right movements). The input is placed on the first tape.
- **Nondeterministic TM**: A Turing machine whose transition function can actually be a relation. That is, for a given state and input symbol, there may be more than one transitions. Naturally, the behavior of such a machine would branch whenever it encounters a multiple-transition point. Note: Recall the distinction between NFA and DFA.

To show the equivalence, we will need to discuss the both directions of simulation. First, a standard TM is reducible to a multiple-tape TM or a nondeterministic TM, i.e., we can simulate a standard TM with a multiple-tape TM or a nondeterministic TM. That is, these new variants are at least as powerful as the standard one. We will discuss the other direction in class.

**Task**: Choose one of the following options and concisely discuss the essence of how the variant of your choice can be simulated by a standard TM. Be prepared to discuss in class.
- Option A: Multiple-tape TM
- Option B: Nondeterministic TM

## Part 3: Alternative Abstract Models of Computation

Although the Turing machine is the most widely used abstract model of computation, there are many others. In this part, we discuss Unlimited Register Machines (URM) and "recursive functions." Note that "recursive function" here refers to a full specification of a computational model, not just instances of recursive calls. Brief descriptions for these models are available in Appendices below. Again, this part is about the question of equivalence, this time between TMs and these new models. If all of these models are equivalent, there must be something universal about the notion of "computation."

For this part, let us limit to one direction, i.e., simulation of these new models by a TM (or reduction of the models to TMs). Then, we know that the TM is at least as powerful as these models.

**Task**: Choose one of the following options and concisely discuss the essence of how the model of your choice could be simulated by a TM. Be prepared to discuss in class.

- Option A: Unlimited Register Machines (URM) [Probably, this option is more intuitive.]
- Option B: Recursive Functions

Survey: Time spent between classes: ______________

# Appendix A: Unlimited Register Machine (URM)

URM is an abstract model of computation, which is closer to a CPU than a TM, with the following properties:

1. There are an infinite number of *registers* called $R_1$, $R_2$, $R_3$, ..., each of which contains a natural number. The number in register $R_i$ is referred to as $r_i$.
2. Register contents may be changed by the following *instructions*:
   - Zero: Z($i$) will reset $r_i$ to 0.
   - Successor: S($i$) will add 1 to $r_i$.
   - Transfer: T($i$, $j$) will copy $r_i$ to $R_j$, i.e., $r_j = r_i$.
   - Jump: J($i$, $j$, $n$) will first evaluate if $r_i = r_j$. If true, the URM will proceed to the $n$th instruction. Otherwise, it will proceed to the next instruction.
3. A *program* consists of a finite sequence of instructions. Execution of a program begins with the first instruction and continues to the next ones until there is no instruction (except for possible jumps).
4. One may assume arbitrary initial values in the registers.

References


- Shepherdson, J. C. and Sturgis, H. E. 1963. Computability of recursive functions. *J. ACM* 10, 217-55. [original]
- Cutland, N. J. 1980. *Computability: An Introduction to Recursive Function Theory*. Cambridge Univ. Press. [contains a description and the use of URM]


# Appendix B: Recursive Functions

The class of recursive functions (μ-recursive functions) is the class of total functions that can be built up from the basic functions (below) by a finite number of operations of *substitution*, *recursion*, and *minimalization* (below):

**Basic functions**

1. Zero: 0 (i.e., returns 0 as constant)
2. Successor: $x + 1$ (i.e., add 1)
3. Projection: $U^n_i(x_1, x_2, ..., x_n) = x_i$ (i.e., picks up the $i$th element from a list of $n$ elements)

**Operations**

- Substitution (composition): Given computable functions $f(y_1, ..., y_k)$ and $g_1(x)$, ..., $g_k(x)$, define $h(x) = f(g_1(x), ..., g_k(x))$
- Recursion: Given computable functions $f(x)$ and $g(x, y, z)$, define
  - $h(x, 0) = f(x)$
  - $h(x, y + 1) = g(x, y, h(x, y))$
- Minimalization: Given computable function $f(x, y)$, define $h(x)$ = the least $y$ such that $f(x, y) = 0$

References


- Godel-Kleene. 1936. [original, not easily accessible]
- Cutland, N. J. 1980. *Computability: An Introduction to Recursive Function Theory*. Cambridge Univ. Press. [detailed discussion of recursive functions]


// End

# Questions/Review

- Halting and other problems
- Reduction/equivalence



# Unit B5: Overview

- Review (based on summary questions)
- Explore the notion of “computation” via the Church-Turing thesis
  - Discuss equivalence of different mechanisms [time permitting]
- Preview Module B Evaluation Workshop



Part 1 Review

# Language

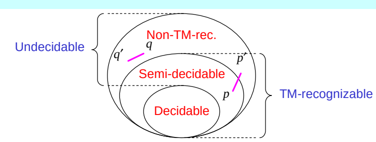
- Formal definition
- TM and language
  - A TM “recognize” a language.
  - A TM “decides” a language.
- Language as a problem
- Countability
  - The set of all the strings
  - The set of all the languages

Examples?



Review

# “Co”-X




- Definition
- Each case
- Proving the theorem: Decidable ⇔ TM-recognizable *and* co-TM-recognizable



Review

# Halting Problem

- Subproof: $ACCEPT$ is undecidable.
  - General idea
  - Use of $L_d$
- Alternative: Not using $L_d$



# Liar Paradox, Revisited

- “This sentence is false.”
- Intuition behind this problem
  - The set of sentences are countable.
  - A sentence in this set can be indexed.
  - The use of “this” can be interpreted as the use of its own index (self reference).
  - The language must have a means for a sentence to refer to an arbitrary sentence (universality) including itself.

Correction

## Reduction

Incorrect

- If $ACCEPT_{TM}$ is reducible to $HALT_{TM}$ and $HALT_{TM}$ is decidable, $ACCEPT_{TM}$ is also decidable.
- If $HALT_{TM}$ is reducible $ACCEPT_{TM}$ to and $HALT_{TM}$ is undecidable, $ACCEPT_{TM}$ is also undecidable.

Correct: reduction with respect to computability analysis

- If $ACCEPT_{comp}$ is reducible to $HALT_{comp}$ and $HALT_{TM}$ is decidable, $ACCEPT_{TM}$ is also decidable.
- If $HALT_{comp}$ is reducible $ACCEPT_{comp}$ to and $HALT_{TM}$ is undecidable, $ACCEPT_{TM}$ is also undecidable.



Correction

## Reduction

Incorrect

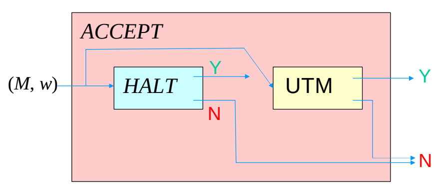
- $ACCEPT_{TM}$ and $NACCEPT_{TM}$ are equivalent.
- $ACCEPT_{TM}$ and $HALT_{TM}$ are equivalent.

Correct

- $ACCEPT_{comp}$ and $NACCEPT_{comp}$ are equivalent.
- $ACCEPT_{comp}$ and $HALT_{comp}$ are equivalent.

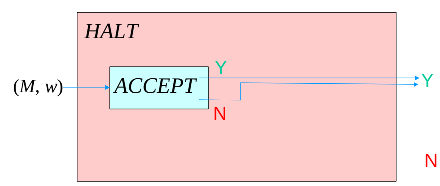


## $ACCEPT_{comp}$ reduces to $HALT_{comp}$






## $HALT_{comp}$ reduces to $ACCEPT_{comp}$






Review

## Rice's Theorem

- Non-trivial properties of languages/TMs are undecidable $\Leftarrow$ Uncountability
  - Note: Trivial properties (e.g., emptiness) that do not divide the entire set is trivially decidable.
- Potentially useful for Ex B5 (Comprehensive Ex)

Text Section 9.3.3



## Other Questions/Comments

- General understanding
- Big picture
- Exam
- More "concrete" problems

Part 2

## Church-Turing Thesis

- Hypothesis as definition
  Computation/effective procedure [informal]
  = TM operation [formal]
- A variety of models of computation are equivalent to TM.
- All sorts of (algorithmic) computation are created equal [and do the same thing, computation].



Take-home exercises

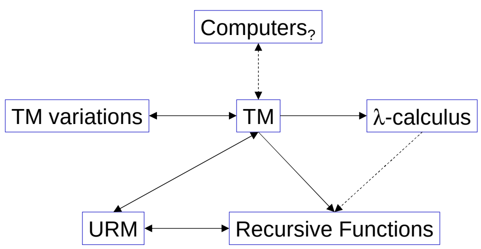
## Connections

- TMs (including variants)
- Computers
  – Different programming paradigms
- Unlimited Register Machine (URM)
- Recursive functions
- λ-calculus [behind LISP]



Preview

## Equivalence Summary






## Proof of Equivalence

- Equivalence of models *M* and *N*
  – *N* simulates *M* and *M* simulates *N*
- Set identity, e.g., $A = B$
  – $A \subseteq B$ and $B \subseteq A$
- Equivalence of propositions, e.g., $p \leftrightarrow q$
  – $p \rightarrow q$ and $q \rightarrow p$
- Equality of values, e.g., $x = y$
  – $x \le y$ and $y \le x$



## TM Variations

- Both accept and reject states (as well as other dying states)
- Semi-infinite tape
- Multiple tapes
- Nondeterministic transition
  – I.e., δ as a relation, not function



## URM (1)

| $R_1$ | $R_2$ | … | $R_i$ | … |
|---|---|---|---|---|
| $r_1$ | $r_2$ | … | $r_i$ | … |

- Instructions:
  – Zero: $Z(i) \Rightarrow r_i \leftarrow 0$
  – Successor: $S(i) \Rightarrow r_i \leftarrow r_i + 1$
  – Transfer: $T(i, j) \Rightarrow R_j \leftarrow r_i$
  – Jump: $J(i, j, n) \Rightarrow$
    if $r_i = r_j$: proceed to the *n*th instruction
    otherwise: proceed to the next instruction

## URM (2)

- Program: A finite sequence of instructions. Sequential execution of instructions except for possible jumps
- Initial values: Specific values can be assumed in the registers.
- Convention: The output in $R_1$.

Simulation by a TM?



## Recursive Functions (1)

- Built up from the basic functions by a finite number of operations of substitution, recursion, and minimalization:
- Basic functions
  - Zero: $0$
  - Successor: $x + 1$
  - Projection: $U^n_i\,(x_1, x_2, ..., x_n) = x_i$



## Recursive Functions (2)

- Operations
  - Substitution (composition): Given computable functions $f(y1, ..., yk)$ and $g_1(x), ..., g_k(x)$, define $h(x) = f(g_1(x), ..., g_k(x))$
  - Recursion: Given computable functions $f(x)$ and $g(x, y, z)$, define
    - $h(x, 0) = f(x)$
    - $h(x, y + 1) = g(x, y, h(x, y))$
  - Minimalization: Given computable function $f(x, y)$, define $h(x)$ = the least $y$ such that $f(x, y) = 0$

Simulation by a TM?



## TM as Function

A TM can simulate a function. When accepts

- Function on strings
  - Input: A list of strings on the tape
  - Output: A string on the tape
- Function on natural numbers
  - Input: A list of natural numbers on the tape
  - Output: A natural number on the tape



## λ-calculus

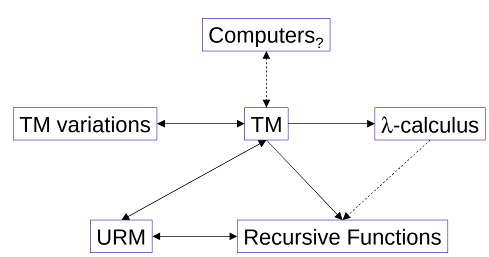
Formal representation of functions

- Abstraction: $\lambda x.exp$
- Application: $exp_1\ exp_2$
- β-reduction: $(\lambda x.exp)\ y$
  - The result is $exp$ where the instances of $x$ is replaced with $y$.
  - E.g., $(\lambda x.f(f(x)))\ 2 = f(f(2))$ Is 2 a function?

In real life?



## Equivalence Summary



## TM vs. Computers

Equivalent?



## Philosophy of Mind

- Hypothesis: Mind is a computer [Fodor: specifically uses TMs as the model].

Implications (if correct)?
Correctness?



## Unit Summary

- Review
- Church-Turing thesis
  - Defines the informal notion of computation
  - Leads to the equivalence of various algorithmic computational approaches



## Computability Summary

- Main goal: To be able to analyze the computability class and identify what we can do with algorithms [Comprehensive Ex]
- Topics
  - Problem as a set
  - Turing machines and (un)decidability
  - Mathematical tools: diagonalization, power set
  - Halting and other main problems; Rice's theorem
  - Reduction
  - Church-Turing thesis



Preview

## Module B Evaluation Workshop

- Required materials (hardcopy) **print in advance**
  - Module B evaluation form
  - Supporting notes  Significance of supp. notes?
  - Exercises
- Activities
  - Module review exercise
  - Peer discussion
  - Reflection and self-evaluation



## Summary Question

- How far have you been thinking Exercise B5 (Comprehensive Exercise)? Explain.
- Questions/Comments/Suggestions

Name: ______________________

# Exercise B5 (Module B Comprehensive Exercise), 2/25/05

Note: This exercise is announced in advance so that we can think about it through this module. The due date of this exercise is the date of Module B Evaluation Workshop.

## Part 1: Analyzing Computability

One of the main goals of Modules A & B (and the Theory of Computability as a whole) is to be able to identify the computability class of real-world problems (so that we won't write a general program for an undecidable problem). In this module, we have been analyzing the main problems such as the halting problem with respect to their universality, self-reference, countability, reduction, etc. The real test for us is whether we can develop and describe our thought process involved in such activities and apply it to a new problem. While this is a major problem in the Theory of Computation, it is unlikely that you will easily find an answer in the literature (what are computer scientists doing?). By all means, it is a challenge. So, you are not expected to come up with a rock solid answer, not to mention a correct answer. Do your best. As usual, you are encouraged to discuss with other students and the instructor (but your writing must be your own, reflecting your own thought, which must be unique to yourself). If we as a class come up with useful analytical tools, it would be great.

Note: Since **Tasks 1** & **2** are closely related, you should tackle them in parallel, or in a circular manner (back and forth).

**Task 1**: Develop your own "heuristics" to analyze a given problem with respect to the computability classes, i.e., decidable, semi-decidable, or non-TM-recognizable. By "heuristics," we mean a speculative, but reasonable method of analyzing *any* given problem. For example, one might consider a rule such as: (i) if self-reference is involved, the problem is undecidable, or (ii) if the problem is uncountable, it is undecidable (these may or may not be true or even relevant; they are just examples of how to phrase a component of your method).

**Task 2**: This task is an application of the previous one. (A) Identify some unique practical problem (in or outside computer science) which would be a good example of demonstrating your heuristics in **Task 1**. (B) Then, provide the set representation of the problem. (C) Finally, apply your heuristics developed in **Task 1** to the problem to analyze its computability class.

## Part 2: Evaluation Form and Supporting Notes

Review the evaluation procedure. Then, complete your evaluation form and supporting notes. Bring them to the evaluation workshop (hard copy). Print them well in advance so that you can avoid potential problems, e.g., not being able to print just before the evaluation.

Survey: Time spent between after Unit B5 before the evaluation workshop: ______________

// End

Name: ______________________

# Module B Review Exercise, 3/1/05

(1) Do the required problems *and* (2) analyze and explain the **computability class** (e.g., decidability) of <u>one</u> of the optional problems, applying your own heuristics. In case your own problem in the comprehensive exercise is similar to any of the optional problems, you must try a different problem. Write your response on back or on another sheet of paper. Note: You are **<u>not</u>** supposed to solve any of the problems.

## Required Problems: '$' Detection

Analyze and compare the computability classes of the following two problems:

A. To find out whether a given string contains at least one instance of the symbol '$'

B. To find out whether a given TM decides on Problem A (above)

## Option 1: Virtual Memory Paging (Architecture)

To come up with an algorithm to find out whether page fault is avoidable, given a program, memory specification, and the specification of the applicable machine architecture

## Option 2: Deadlock Prevention (OS)

To come up with an algorithm to prevent deadlocks, given a set of processes (which may spawn subprocesses) and resources (which may become unavailable)

## Option 3: Optimization (Compilers)

To design a procedure to generate the most optimized target code of the given source code with respect to the target code size

## Option 4: Protocol Security (Networks)

To develop a process to find out whether a given network protocol is secure

## Option 5: Semantic Errors (Databases)

To identify a mechanism to detect semantic errors (e.g., contradictory queries, which would return an empty set) in database queries

## Option 6: Planning (AI)

To develop a program to construct a plan (of actions) that would lead to the desired goal state, given a set of primitive actions

## Option 7: Best Solution

To find the best solution, given a problem and some specification about the goodness of a solution

## Option 8: Decidability Classes

To identify the computability class of a given problem

// End

Name: ______________________

# Exercise B6/C0, 3/1/05

This exercise is due on Tue., Mar. 15, 2005 (i.e., after the spring break).

## The Computational Power of Turing Machines

We have been using the Turing machine as our model of computation. It is convenient because with TMs, we can capture all of (and exactly) the algorithms as we informally understand (Church-Turing thesis). We also discussed that nontrivial properties of TMs are undecidable, i.e., not captured by the TM (Rice's Theorem). In addition to the main problems, we noted more problems in this category in Module B Review Exercise. So, we know that TMs are limited, esp. when we need to deal with, say, practical questions involving "big" issues in the CS industry. On the other hand, we can easily imagine situations where even the TM seems to be overkill. For example, to check whether or not a given string is an acceptable variable name (e.g., in a compiler), we do not need the full power of the TM (i.e., no need for the infinite tape). If we can solve a problem with an easier means, we should use it instead of unnecessarily complicated means ("principle of parsimony" and "Ockham's razor"). The ability to identify and use the minimal mechanism for a given problem is essential in virtually any area of study or work. Do you have this skill? This is another property that would distinguish computer scientists from programmers. "Elegant" solutions are often sought not because of esthetics but because of practicality, e.g., for easy understanding and maintenance.

**Task**: Write a concise essay about whether or not the TM is an appropriate model *with respect to its computational power*, referring to (i) your problem from Exercise 00 and (ii) your mini research question(s) from Exercise A6/B0 or different one(s) from the list (below). In addition to discussing these problems in their entirety, you must also consider possible subproblems (cf. **Note 1** below). Apply the skills learned in earlier CS courses, e.g., object orientation and modular programming, and analyze whether any of the problems/subproblems would need the full power of the TM.

**Note 1**: We know that compilation is a decidable problem. Thus, a TM is surely sufficient for the problem. But as we briefly discussed in class, a compiler typically consists of multiple phases/modules. When these phases are considered as subproblems, they would require mechanisms of different complexity. For example, the lexer requires a DFA (but not a full TM or even a stack), the parser requires a stack (but not a full TM), and semantic analysis (e.g., variable reference) requires some sort of table (may correspond to a TM). This way, when we deal with a simpler problem, e.g., lexical analysis, we can do it as easily/fast as possible. In a similar manner, your problem(s) may be broken into subproblems, which may require different mechanisms of different complexity.

**Note 2**: For each of the research questions listed below, there is some connection to the Theory of Computation. Although algorithmic computation is limited as we observed in Modules A & B, being able to identify such limitation could let us pinpoint the source of complexity beyond the traditional computation. With this ability, one will be able to learn a broad range of subjects in a systematic manner. When I first studied the Theory of Computation on my own almost two

decades ago (no CS courses taken and no plan to study CS at that point), the motivation was to understand the mechanism behind human language (i.e., linguistics). But the Theory certainly changed the way I understand human language (well, at least certain systematic aspects of it). When I started my graduate work in CS and studied CS topics such as programming languages, compilers, etc., I felt "OK." But when I studied cognitive science and complex systems, it was the limitations of the Theory of Computation that I appreciated. It may be difficult to "use" the Theory in this manner, esp. during this semester (we do not really discuss much, outside the Theory topics in this course). But I hope you keep this in mind and try to apply the acquired ideas throughout your career. Whether you can connect apparently different things and analyze them systematically would make a huge difference in your life, regardless of your career choice.

### List of sample research questions from Exercise A6/B0

1. Can **organizational dynamics** be modeled as an algorithm?
2. Can **evolution** be modeled as an algorithm?
3. Can **ecology** be modeled as an algorithm?
4. Can **human development** be modeled as a computer?
5. Can our **minds** be modeled as a computer?
6. Can **vision** be modeled as an algorithm?
7. Can **learning** be modeled as an algorithm?
8. What would be the minimal mechanism to process **human language**?
9. Can the entire situation of an arbitrary **game** be modeled as an algorithm?
10. Would "perfect" **user modeling**, e.g., for web search, be possible?
11. Would "perfect" **computer security** be possible?
12. Can the entire process of **software engineering** be modeled computationally?
13. Can **computer networks** be modeled as a single computer?
14. Could **biology** be reduced to physics?
15. Could some computer generate real (not pseudo) **random numbers**?
16. Would it be possible to decide whether the given numbers are **random**?
17. Would **randomization** affect computability and/or complexity?
18. Would **parallelism** affect computability and/or complexity?
19. Would **artificial neural network** be more powerful than TMs?
20. Would **cellular automata** be more powerful than TMs?
21. Would the use of **analog** (or fuzzy) values affect computability?
22. Would relativistic, quantum, or some other **modern-physics**-based computation surpass TMs?
23. Can all the cases of **on-line algorithm**s be simulated by off-line computation? [On-line algorithms would obtain inputs as the time progress. Off-line computation would provide all the possibilities as input at once. Cf. function-to-relation conversion used to fold the output within the input]
24. Would **oracle computing** affect computability? [Oracle computing: A TM with the capability to ask questions to another mechanism]
25. Would **persistent TM** be able to compute more than the standard TM? [Persistent TM: Multiple sessions of TM operation with some memory between them]
26. Would **accelerating** a TM give more power?
27. Would slight **error tolerance** affect any aspect of the Theory of Computation?
28. What would be the ability of a finite automaton with a **queue**?
29. What would be the effect of "**constant**" (as in complexity analysis) in practice?
30. Can any **mathematical function** be represented computationally?
31. What exactly are **power sets** doing to the Theory of Computation?
32. Is what you can do in **logic** the same as what you can do with computation?
33. If you have a research question of **your own**, please consult the instructor first.

// End

# Unit B6 Supplement, 3/4/05

## Ex B5 Compiled Heuristics

Here is a (edited and expanded) collection of ideas from your Ex B5, classified according to the involved techniques. In some cases, wording is subtle.

**TM Construction** [Useful for simple problems, e.g., combinatory problems]
1. If it is possible to create an algorithm (a terminating TM) to solve the problem, it is decidable.
2. If it is possible to create a TM to "recognize" the problem/language (but not guaranteed to terminate), it is TM-recognizable.

Note: The following cases are insufficient as conditions.
- Not being able to come up with an algorithm/TM (in the future?)
- Potential of looping (too informal; why did we discuss the main problems in detail?)
- Solving at least one input (a single input is not enough)

**Set operation** [Useful when multiple properties are known]
3. If the problem is TM-recognizable and undecidable, it is semi-decidable.
4. If the problem is both TM-recognizable and co-TM-recognizable, it is decidable.
5. If the complement of the problem is decidable, it is decidable.
6. If the complement of the problem is semi-decidable, it is non-TM-recognizable.
7. If the complement of the problem is non-TM-recognizable, it is undecidable. [added]

**Reduction** [Useful when another problem is known]
8. If the problem can be reduced to a known decidable (TM-recognizable) problem, the former problem is decidable (TM-recognizable).
9. If a known undecidable (non-TM-recognizable) problem can be reduced to the problem, the latter problem is undecidable (non-TM-recognizable).
10. If the *computability analysis* of the problem can be reduced to that of a known decidable (TM-recognizable) problem, the former problem is decidable (TM-recognizable). [added]
11. If the *computability analysis* of a known undecidable (non-TM-recognizable) problem can be reduced to that of the problem, the latter problem is undecidable (non-TM-recognizable). [added]

**Countability** [Useful when there are more languages (uncountable) than all of TMs (countable)]
12. If the problem is about a non-trivial property of a TM/language, it is undecidable (Rice's theorem).
13. If the problem is a non-empty, proper subset of an uncountable set (or the power set of a countably infinite set), it is undecidable. [added]

Note: The following cases are insufficient as conditions.
- Uncountable set (there are trivial decidable problems, i.e., constant problems)
- Countably infinite set (cf. $L_d$ is countable)
- Finite set (cf. a truly random problem)

**Other** [Useful for complex representations, e.g., algorithm, function, logic, grammar]

14. If the problem involves a mechanism that allows counting (natural numbers) and self-reference, it is (probably) undecidable. [I am not prepared to do a rigorous justification.]

Note: The following cases are insufficient as conditions.

- Universality or self-reference (this could happen in a limited domain which cannot count)

// End

Name: ______________________

# Module B Extra Problems, 3/4/05

This is an *optional* problem set. These are provided for those who want to try some textbook-style problems. The problems are actually taken from a classic textbook. If you want to discuss these problems, contact the instructor.

**X1**. Consider a model of a Turing machine in which each move permits the read-write head to travel more than one cell to the left or right, the distance and direction of travel being one of the arguments of $\delta$. Give a precise definition of such an automata and sketch a simulation of it by a standard Turing machine. [Note: $\delta$ is the transition function.]

**X2**. Show that the following problem is undecidable. Given any Turing machine $M$, $a \in \Gamma$, and $w \in \Sigma^+$, determine whether or not the symbol $a$ is ever written when $M$ is applied to $w$. [Note: "$\Sigma^+$" is a regular expression, i.e., a sequence of one or more symbols in the set $\Sigma$.]

**X3**. Show that the following problems are undecidable:
A. $L(M)$ contains any string of length five.
B. $L(M)$ is regular.
[Note: $L(M)$ denotes the language accepted by a TM $M$.]

**X4**. Show that there is no algorithm for deciding if any two Turing machines $M_1$ and $M_2$ accept the same language. How would your conclusion be affected if $M_2$ is a finite automaton?

**X5**. Let $B$ be the set of all Turing machines that halt when started with a blank tape. Show that this set is recursively enumerable, but not recursive. [Note: Review the terms.]

**X6**. Let $M_1$ and $M_2$ be arbitrary Turing machines. Show that the problem "$L(M_1) \subseteq L(M_2)$" is undecidable.

**X7**. Determine whether or not the following statement is true: Any problem whose domain is finite is decidable.

**X8** [**Post's Correspondence Problem** (**PCP**)]. Consider a non-empty set of pairs of strings $S = \{(x_1, y_1), (x_2, y_2), ..., (x_k, y_k)\}$. The problem is to find some sequence of members of $S$ such that the concatenation of the $x$ strings are identical to that of $y$ strings. $\Pi x_i = \Pi y_i$ where $\Pi$ denotes the concatenation of strings and $i$ denotes the index of a string. Show that PCP is undecidable. [Note: Do web search for more details about this well-known problem.]

// End

# Module B Evaluation

Review your portfolio

- My comments are sporadic and scattered on Review Ex, Comprehensive Ex, Supp. Notes, and Reflective Essay.
- You are encouraged to clarify and discuss my comments.
- You can keep the folder till the next class; then, return it to me.



# Unit B7: Overview

- Review
  - Comprehensive/Review Exercises, etc.
- Explore the notion of "computation" via the Church-Turing thesis
  - Discuss equivalence of different mechanisms



# Correction on Eval Form B

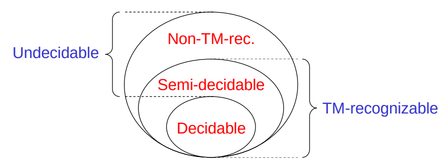
4. The algorithmic notion of computation can be represented in a variety of equivalent forms, which define a bounded class of sets. That is, there is a limit to algorithmic computation. [**computability**]
   c. Can explain logically that the halting problem is semi-decidable and infinite-loop detection is ~~unsolvable~~ non-TM-recognizable.



Review

# Computability Classes

- With respect to impossibility, problems can be classified into 3 disjoint sets, based on whether a TM exists and/or always gives an answer.






## Module B Comprehensive/Review Exercise

- Required Problems: '$' Detection
  - **A.** To find out whether a given string contains at least one instance of the symbol '$'
  - **B.** To find out whether a given TM decides on Problem A
- Option 1: Virtual Memory Paging (Architecture)
- Option 2: Deadlock Prevention (OS)
- Option 3: Optimization (Compilers)
- Option 4: Protocol Security (Networks)
- Option 5: Semantic Errors (Databases)
- Option 6: Planning (AI)
- Option 7: Best Solution
- Option 8: Decidability Classes



# Church-Turing Thesis

- Hypothesis as definition
  Computation/effective procedure [informal]
  = TM operation [formal]
- A variety of models of computation are equivalent to TM.
- All sorts of (algorithmic) computation are created equal [and do the same thing, computation].

Take-home exercises

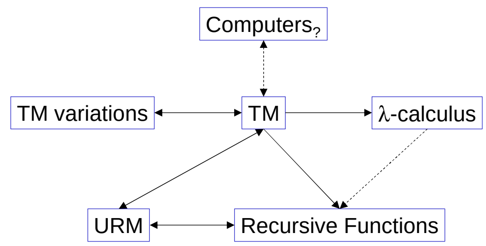
# Connections

- TMs (including variants)
- Computers
  - Different programming paradigms
- Unlimited Register Machine (URM)
- Recursive functions
- $\lambda$-calculus [behind LISP]



Preview

# Equivalence Summary






# Proof of Equivalence

- Equivalence of models *M* and *N*
  - *N* simulates *M* *and* *M* simulates *N*
- Set identity, e.g., $A = B$
  - $A \subseteq B$ and $B \subseteq A$
- Equivalence of propositions, e.g., $p \leftrightarrow q$
  - $p \to q$ and $q \to p$
- Equality of values, e.g., $x = y$
  - $x \le y$ and $y \le x$

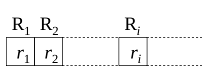


# TM Variations

- Both accept and reject states (as well as other dying states)
- Semi-infinite tape
- Multiple tapes
- Nondeterministic transition
  - I.e., $\delta$ as a relation, not function



# URM (1)




- Instructions:
  - Zero: $Z(i) \Rightarrow r_i \leftarrow 0$
  - Successor: $S(i) \Rightarrow r_i \leftarrow r_i + 1$
  - Transfer: $T(i, j) \Rightarrow R_j \leftarrow r_i$
  - Jump: $J(i, j, n) \Rightarrow$
    if $r_i = r_j$: proceed to the $n$th instruction
    otherwise: proceed to the next instruction



# URM (2)

- Program: A finite sequence of instructions. Sequential execution of instructions except for possible jumps
- Initial values: Specific values can be assumed in the registers.
- Convention: The output in $R_1$.

Simulation by a TM?

# Recursive Functions (1)

- Built up from the basic functions by a finite number of operations of substitution, recursion, and minimalization:
- Basic functions
  - Zero: $0$
  - Successor: $x + 1$
  - Projection: $U^n_i\ (x_1, x_2, ..., x_n) = x_i$



# Recursive Functions (2)

- Operations
  - Substitution (composition): Given computable functions $f(y1, ..., yk)$ and $g_1(x), ..., g_k(x)$, define $h(x) = f(g_1(x), ..., g_k(x))$
  - Recursion: Given computable functions $f(x)$ and $g(x, y, z)$, define
    - $h(x, 0) = f(x)$
    - $h(x, y + 1) = g(x, y, h(x, y))$
  - Minimalization: Given computable function $f(x, y)$, define $h(x)$ = the least $y$ such that $f(x, y) = 0$

Simulation by a TM?



# TM as Function

A TM can simulate a function. When accepts

- Function on strings
  - Input: A list of strings on the tape
  - Output: A string on the tape
- Function on natural numbers
  - Input: A list of natural numbers on the tape
  - Output: A natural number on the tape



# λ-calculus

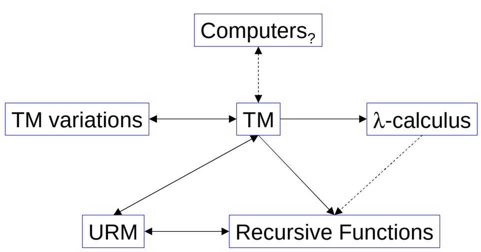
Formal representation of functions

- Abstraction: $\lambda x.exp$
- Application: $exp_1\ exp_2$
- β-reduction: $(\lambda x.exp)\ y$
  - The result is $exp$ where the instances of $x$ is replaced with $y$.
  - E.g., $(\lambda x.f(f(x)))\ 2 = f(f(2))$ Is 2 a function?

In real life?



# Equivalence Summary






# TM vs. Computers

Equivalent?

## Philosophy of Mind

- Hypothesis: Mind is a computer [Fodor: specifically uses TMs as the model].

Implications (if correct)?
Correctness?



## Unit Summary

- Review
- Church-Turing thesis
  - Defines the informal notion of computation
  - Leads to the equivalence of various algorithmic computational approaches



## Computability Summary

- Main goal: To be able to analyze the computability class and identify what we can do with algorithms [Comprehensive Ex]
- Topics
  - Problem as a set
  - Turing machines and (un)decidability
  - Mathematical tools: diagonalization, power set
  - Halting and other main problems; Rice's theorem
  - Reduction
  - Church-Turing thesis

# Unit B7 Supplement, 3/7/05

## Part 1: Church-Turing Thesis

As we discussed in class, the Church-Turing thesis states that the informal notion of computation is captured by Turing machines. Although it is called a "thesis" (i.e., proposition), it involves an informal notion, which can never be "proven." That is why, it should be considered more as a definition. However, due to the equivalence between Turing machines with a wide range of other formalisms, most TM variants, real computers (with qualifications), Unlimited Register Machines, recursive functions, lambda calculus, etc., this thesis indeed captures the essence of at least algorithmic part of computation (more about non-algorithmic computation later). That is, if there are so many different ways to represent the same notion, albeit informal, such a notion must have something deep. For example, through the connection between TMs and recursive functions, we can often translate results in computer science and mathematics back and forth, providing different perspectives useful in different contexts.

Here are examples of some other abstract models (see the definitions in Exercise B4 and Unit B5/B7). If you skim the examples below, you may realize why the TM is the most popular model of computation.

**Unlimited Register Machine (URM)**

- Example: Addition $r_1 + r_2$
  - Reset $r_3 = 0$, with *Zero*
  - Count up $r_1$ and $r_3$ until $r_3 = r_2$ , iterating the basic operations including *Successor* and *Jump*
  - The result of the addition is in $r_1$.

The simulation of a URM with a TM is fairly intuitive. The other direction requires a few tricks. For example, we will need to be able to handle all symbolic computation with URMs; this can be done by associating all the tape symbols with unique natural numbers (ASCII code?). A URM can then represent the TM tape within its register, encode the tape position in a special register, simulate the basic TM operations as subroutines, etc.

**Recursive Functions**

- Example: Addition $add(x, y)$
  - $add(x, 0) = x$
  - $add(x, y + 1) = Successor(add(x, y))$ [using the basic function *Successor*]
- Example: Multiplication $mult(x, y)$
  - $mult(x, 0) = 0$
  - $mult(x, y + 1) = add(mult(x, y), x)$ [using pre-defined functions]
- Example: Factorial $fac(x)$
  - $fac(0) = 1$
  - $fac(y + 1) = mult(fac(y), Successor(y))$ [using pre-defined functions]

The simulation of a recursive function with a TM is fairly intuitive. The other direction can go indirectly through the simulation of a URM with a recursive function. This simulation is still

involved. First, it is not very intuitive to simulate certain URM operations with a recursive function (e.g., Jump as a recursive function?). One way to cope with this situation is to limit the simulation to the cases where the given URMs that can compute a function (if not, we can implementing a looping function). So, by this assumption, we know that the URM (program) $P$ receives inputs $\boldsymbol{x} = x_1, x_2, ..., x_k$ (boldface for a list or "vector") in the respective registers and computes the result $P(\boldsymbol{x})$ to be left in $R_1$. Let us define two functions. First, $result(\boldsymbol{x}, t) = r_1$ computing the result. Second, $next(\boldsymbol{x}, t)$ would return 0 if the computation terminates, or would return the next step (e.g., a function ID) if the computation still continues. If a function is defined, $P(\boldsymbol{x})$ must terminate at some step $t_i$ (i.e., $t_i$ = the least $t$ such that $next(\boldsymbol{x}, t) = 0$), and the result $result(\boldsymbol{x}, t_i)$ appears $R_1$. Otherwise, $P(\boldsymbol{x})$ would loop (i.e., $next(\boldsymbol{x}, t) \neq 0$ for any $t$). In either case, the desired function (which is computed by $P$) is defined as a recursive function $result(\boldsymbol{x}$, the least $t$ such that $next(\boldsymbol{x}, t) = 0)$

**Lambda Calculus**

- Example: Identity function
  - $\lambda x.x$
- Example: Church numeral
  - Representation of a natural number $n$: $\lambda f.\lambda x.f^{\,n}(x)$
  - $0 = \lambda f.\lambda x.x$ [Note: $\lambda f.\lambda x.f^{\,0}(x)$]
  - Succ $= \lambda n.\lambda f.\lambda x.(f\ ((n\ f)\ x))$
  - $1 = \text{Succ } 0 = \lambda f.\lambda x.(f\ (((\lambda f.\lambda x.x)\ f)\ x)) = \lambda f.\lambda x.(f\ (\lambda x.x\ x)) = \lambda f.\lambda x.(f\ x)$
    [Note: $\lambda f.\lambda x.f^{\,1}(x)$]
- Example: Conditional
  - T $= \lambda x \lambda y.x$ [i.e., picking the first argument]
  - F $= \lambda x \lambda y.y$ [i.e., picking the second argument]
  - B M N (if B then M else N) where B is T or F

The simulation of lambda calculus with TMs would be intuitive, although tedious as usual. The other direction can go through recursive functions.

# Part 2: Rice's Theorem

Although you may not have noticed, many of the problems in Module B Review Exercise (Required Problem B, Options 1 through 6) can be answered very concisely by using Rice's theorem. As the option problems indicate, many general problems in computer science refers to a non-trivial property of a certain general procedure. Thus, this theorem is widely applicable and useful to pinpoint the limitations of algorithmic computation. This is a review of this theorem with some insight into why this theorem holds. First, the definition again.

**Rice's Theorem**: A non-trivial property of a TM is undecidable.

Note that a "non-trivial" property refers to any property of a TM that is not "constant" in the following sense. If the problem is to return "yes" or implement a constant function (e.g., *Zero*), the behavior is fixed and always decidable. Such a constant behavior is considered "trivial." Everything else, including very simple problem such as telling if the input has at least one '$' (Comprehensive Exercise Required Problem A), is non-trivial. We also know that each TM is

associated with a language which the TM at least "recognizes" (but not necessarily decides). The operation of a TM can also be associated with an algorithm (if terminates), procedure, etc. So, we might consider the following corollary as a slightly more applicable form of Rice's theorem.

**Corollary**: A non-constant property of a TM, language, algorithm, procedure, recursive function, etc. is undecidable.

Let us see why this is the case, using Comprehensive Exercise Required Problem B, defined as $TM_{DOLLAR} = \{M \mid \text{TM } M \text{ accepts a string that contains at least one '\$'}\}$. First this is a problem about TMs (cf. Required Problem A, which is a problem about strings). Although it is not explicitly indicated as in $ACCEPT_{\text{TM}} = \{\langle M, w\rangle \mid \text{TM } M \text{ accepts } w\}$, $TM_{DOLLAR}$ still needs to pair the given TM with all sorts of string inputs. But the sequence of strings given to the TM, interpreted as a language, may be a non-TM-recognizable one. Then, regardless of the TM at work at that point, such an input cannot be recognized. So, this problem must be undecidable. In a similar fashion, we can intuitively understand that Rice's theorem and the corollary above, referring to any nontrivial property of a TM.

For this particular problem $TM_{DOLLAR}$, we can narrow down the computability class further. Since it would be possible to create a modified UTM that simulates the given TM on an input. If the modified UTM evaluates that the given TM detects at least one '$', we can pick up all the positive cases. Thus, it is semi-decidable, as in $ACCEPT_{\text{TM}}$ and $HALT_{\text{TM}}$.

// End

# Module B Evaluation

Review your portfolio

- My comments are sporadic and scattered on Review Ex, Comprehensive Ex, Supp. Notes, and Reflective Essay.
- You are encouraged to clarify and discuss my comments.
- You can keep the folder till the next class; then, return it to me.



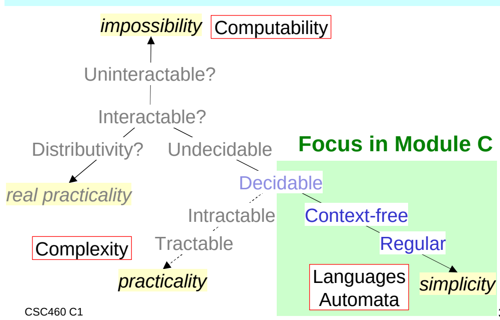
# Unit C1: Overview

- Overview Module C
- Discuss Ex B6/C0 “The Power of TMs”
- Introduce another model of computation (equivalent to the TM)
- Explore the effects of downgrading TMs
- Identifying the minimal mechanism for the given problem
- Preview Exercise C1 “Chomsky Hierarchy”

rough intro; more details later



# Overview: Theory of Computation






# Module C Goals (1)

**Content Goals**

5. Formal languages and automata
   a. “Hierarchy” of TM-recognizable languages
   b. Unrestricted grammars (rewriting system) ~ TMs
   c. Context-free languages (CFLs) ~ CFGs ~PDAs). When/how to use this class to analyze problems.
   d. Deterministic subset of CFLs ~ DPDAs
   e. Regular languages (regular sets~ RegExps~ FSAs). When/how to use this class to analyze problems.
   f. Show that some language is *not* regular.
   g. Show that some language is *not* context-free. That certain properties of CFLs are undecidable.
   h. Could explain this goal to CS students outside this class.



# Module C Goals (2)

**Content Goals**

7. Power set
   a. Nondeterminism as the power set of the possible states.
   b. Nondeterminism with respect to TMs, PDAs, FSAs.

**Performance Goals**

4. Critical attitude
   a. *Critically* analyzed the usefulness of the “languages/automata” area of the (traditional) Theory of Computation
5. Communication
   a. Completed all the exercises on time (take-home and in-class).
   b. Made conscious efforts to promote transfer of learning among students

choosing the minimal mechanism

different background for this module



# Ex B6/C0: The Power of TMs

The entire problem

- TMs are the appropriate mechanism.
- TMs are too weak.
- TMs are too powerful.

Modules/subproblems

## Power-Economy Tradeoff

- Complex mechanisms are more powerful than simple ones (can represent more variety of problems/languages).
- Simple mechanisms are less expensive than complex ones (can operate simply and faster in general).
- An appropriate level would be the least powerful one for the requirement.



## Downgrading TMs

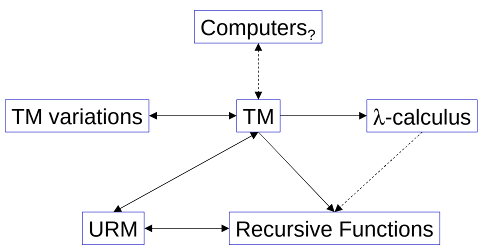
- Imposing a specific tape organization
  - As a stack
  - As a queue
- Limiting the tape size
  - Set some finite fixed bound
  - Limit the growth (e.g., linearly with the input)
- Limiting the head mechanism
  - Read-only
  - One-way scanning only

alternative models



Review

## TM-Equivalent Models




Other models?



## Identifying Alternative Models

- Problem ~ set
  - Most commonly as a language (set of strings)
  - Recognized by a TM
- Whatever mechanism that can recognize (or generate) a string can be examined for the equivalence with a TM (or some downgraded forms).



## Grammar

- Grammar [unrestricted grammar, semi-Thue system]
  - Rewrite rules of the form: $\alpha \to \beta$
    where $\alpha$, $\beta$ are strings of symbols and $|\alpha| > 0$

- $S \to NP_1\ V\ NP_2$
- $NP_1\ V\ NP_2 \to NP_2$ is Ved by $NP_1$
- $NP_1 \to$ liisa | tiina
- $NP_2 \to$ mikko | seppo
- $V \to$ kisses | kicks
- Ved $\to$ kissed | kicked

TM vs. grammar?



## Using a Grammar

- Generating a string
- Accepting a string (parsing)

- $S \to NP_1\ V\ NP_2$
- $NP_1\ V\ NP_2 \to NP_2$ is Ved by $NP_1$
- $NP_1 \to$ liisa | tiina
- $NP_2 \to$ mikko | seppo
- $V \to$ kisses | kicks
- Ved $\to$ kissed | kicked

- Terminal symbols [lower case]: Cannot be rewritten
- Nonterminal symbols [UPPER CASE]: Must be rewritten

## TM vs. Grammar

- Equivalent!
- I.e., both can recognize/generate exactly the same class of languages
  - TM-recognizable
  - Recursively enumerable (RE) [standard term in this context]
- Proof idea
  - A grammar simulated by a TM
  - A TM simulated by a grammar



## TM or Grammar?

- Modeling computation
- Modeling function
- Classifying sets (including computational problems)
- Declarative specification
- Specification of programming languages
- Description of human languages
- Analysis of different classes



## Grammar Variations

- Conditions on the length of LHS (left-hand side) and/or RHS
- Conditions on the type of string on the RHS

- $S \rightarrow NP_1\ V\ NP_2$
- $NP_1\ V\ NP_2 \rightarrow NP_2$ is Ved by $NP_1$
- $NP_1 \rightarrow$ liisa | tiina
- $NP_2 \rightarrow$ mikko | seppo
- $V \rightarrow$ kisses | kicks
- Ved $\rightarrow$ kissed | kicked



## Grammar Hierarchy

- Type 0: Unrestricted Grammar
- Type 1: Context-Sensitive Grammar (CSG)
  - $|LHS| \leq |RHS|$ [i.e., no shrinking]
- Type 2: Context-Free Grammar (CFG)
  - $|LHS| = 1$ [i.e., ignore the context of rewriting]
- Type 3: Regular Grammar ~ Regular Expression
  - RHS has at most one nonterminal at the left/right edge.

examples?



## Automata Hierarchy

- Type 0: TM
- Type 1: Linear Bounded Automaton (LBA)
  - Tape space limited to the input size
- Type 2: Pushdown Automaton (PDA)
  - Tape as a stack (with a separate input tape)
- Type 3: Finite-State Automaton (FSA)
  - Head movement only to the right

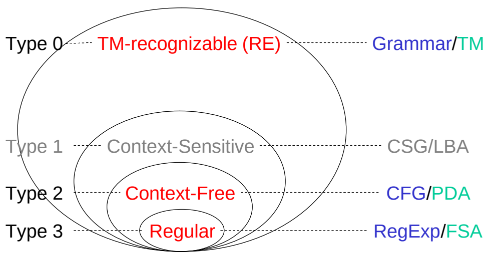
schematics



## Language Hierarchy

- Type $n$ languages are accepted/generated by Type $n$ grammars and recognized by Type $n$ automata.
- Type 0: RE (TM-recognizable)
- Type 1: Context-Sensitive Languages (CSLs)
- Type 2: Context-Free Languages (CFLs)
- Type 3: Regular Languages

## Chomsky Hierarchy




Contributions of Chomsky



## Group Exercise 1

Identify the minimal type for the following

- Pattern matching?
- Programming language specification?
- Representing human language?

Discuss the following

- Effects of nondeterminism?
- Is the hierarchy exhaustive?
  - Where is the set of decidable languages?
  - Class like Type 1.5?

See Ex C1

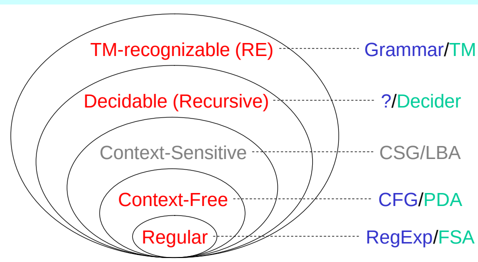


## Recursive Sets

- Decidable = Recursive
- Theorem: CSLs $\subset$ Recursive Sets (i.e., proper inclusion)
- Proof idea
  - CSLs $\subseteq$ Recursive Sets
  - There is a recursive set that is not a CSL.



## Extended Chomsky Hierarchy






## Power-Economy Tradeoff, Revisited

- Higher-level grammars/automata are more powerful than lower-level ones (can represent more variety of languages).
- Lower-level grammars/automata are less expensive than higher-level ones (can operate simply and faster in general).
- An appropriate level would be the least powerful one for the requirement.



## Unit Summary

- Grammars
- Connection between grammars, automata, and languages (problems)
- Chomsky hierarchy
- Identifying the minimal mechanism for the given problem

## Summary Question

- What would be your guiding principle to identify the simplest mechanism/approach (not necessarily in CS)? Explain.
- Questions/Comments/Suggestions

# March 15, 2005: Alternative Class Activities

In case the instructor cannot make the class this day, do the following activities *together* in class:

1. Review your Module B Review Exercise and Exercise B5 heuristics, also using the compiled heuristic and B7 supplement on Rice's theorem (available on-line). The students who were present on Friday, March 4 are invited to share the discussion on that day.
2. Discuss the Church-Turing thesis, also using the B7 supplement (available on-line). Again, the students who were present on Friday, March 4 are invited to share the discussion on that day.
3. Form groups of two (or three students). Discuss your Exercise B6/C0 "The Power of TMs." In particular, compare the entire problem and subproblems with respect to whether you would need the full power of the TM. You should also make reference to the content of the slide below. Then, compare the results altogether.

Power-Economy Tradeoff

- Complex mechanisms are more powerful than simple ones (can represent more variety of problems/languages).
- Simple mechanisms are less expensive than complex ones (can operate simply and faster in general).
- An appropriate level would be the least powerful one for the requirement.

CSC460 C1 1

4. Go over Module C Content and Performance Goals (available on-line; see the eval form in Unit C6). You do not need to print the document (hard copies will be distributed in the next meeting).
5. Form groups of two (or three students). Discuss the effects of the limitations on TMs. Then, compare the results altogether.

Downgrading TMs

- Imposing a specific tape organization
  - As a stack
  - As a queue
- Limiting the tape size
  - Set some finite fixed bound
  - Limit the growth (e.g., linearly with the input)
- Limiting the head mechanism
  - Read-only
  - One-way scanning only

CSC460 C1 alternative models 8

For the next class on Friday, March 28, 2005, write up a report of the in-class activities.

// End

Name: ______________________

# Exercise C1, 3/15/05

## Part 1: Chomsky Hierarchy for Your Problems

The Chomsky hierarchy can help you identify the minimal mechanisms (grammars/automata) that are appropriate for your decidable problem. For example, if you know that parenthesis matching can be specified by a CFG and processed by a PDA, you would use these instead of more powerful and thus more costly unrestricted grammar and TM. At the same time, you know that you cannot use a regular expression or a FSA to do the job. We will be studying the two classes "context-free" and "regular" more in detail in the coming units. In this exercise, you will revisit your own problems, where you may want to review what you learned in other courses (most notably "compilers") and develop some intuition behind the hierarchy.

**Task**: Revisit the problems (including the subproblems) you discussed in Exercise B6/C0. Wherever possible, classify the problems based on the extended Chomsky hierarchy (i.e., TM-recognizable, decidable, context-free, or regular).

## Part 2: Grammar and TM Variants

As we discussed in Module B, all nontrivial properties of a TM are undecidable. That is, we cannot have a general mechanism to decide on them. On the other hand, simpler mechanisms are in general much more manageable. So, in many cases, it makes sense to consider and use simpler mechanisms where appropriate. In this part, you *speculate* the power of some downgraded versions of grammars and TMs. If you have questions, contact the instructor.

**Task**: For each of the following TM/grammar variants, try to find out what class of languages (from the extended Chomsky hierarchy) can be accepted/recognized/generated.

A. CFGs without empty productions (note that CFGs can have rules of the form $A \rightarrow \varepsilon$, where $\varepsilon$ is the empty string)
B. CFGs with rules such that $|\text{RHS}| \leq 2$ (e.g., $A \rightarrow B\ C$ is acceptable, but $A \rightarrow B\ C\ D$ is not)
C. TMs with 2 stacks along with an additional input tape (i.e., like PDAs except that they are equipped with 2 independent stacks)
D. TMs with a queue along with an additional input tape (i.e., like PDAs except that the tape is used as a queue, not a stack)

Hints

- For the first two problems, try to transform an arbitrary CFG to the limited versions. If you can do that, the limited versions can still accept all CFLs. If not, they may be less powerful.
- For Problem C, see whether you can access an arbitrary element in one stack using the second stack (if so, the 2 stacks can simulate a single infinite tape). For Problem D, see whether the queue can simulate a stack or an infinite tape.
- You are not expected to use references. However, if you do, cite them.

Survey: Time spent between classes: ______________

# Unit C1 selected slides for the in-class Group Exercise 1

## Grammar Hierarchy

- Type 0: Unrestricted grammar
- Type 1: Context-Sensitive Grammar (CSG)
  - $|LHS| \leq |RHS|$ [i.e., no shrinking]
- Type 2: Context-Free Grammar (CFG)
  - $|LHS| = 1$ [i.e., ignore the context of rewriting]
- Type 3: Regular Grammar ~ Regular Expression
  - – RHS has at most one nonterminal at the left/right edge.

CSC460 C1 14

## Automata Hierarchy

- Type 0: TM
- Type 1: Linear Bounded Automaton (LBA)
  - – Tape space limited to the input size
- Type 2: Pushdown Automaton (PDA)
  - – Tape as a stack (with a separate input tape)
- Type 3: Finite-State Automaton (FSA)
  - – Head movement only to the right

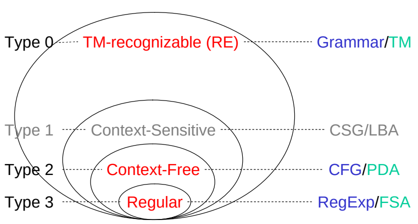
CSC460 C1 15

## Language Hierarchy

- Type *n* languages are accepted/generated by Type *n* grammars and recognized by Type *n* automata.
- Type 0: RE (TM-recognizable)
- Type 1: Context-Sensitive Languages (CSLs)
- Type 2: Context-Free Languages (CFLs)
- Type 3: Regular Languages

CSC460 C1 16

## Chomsky Hierarchy




Contributions of Chomsky

CSC460 C1 17

// End

# Exercise C1

Part 1: Your Own Problems

Part 2: Grammar and TM Variants

A. CFGs without empty productions

B. CFGs with rules such that |RHS| $\leq$ 2

C. TMs with 2 stacks

D. TMs with a queue



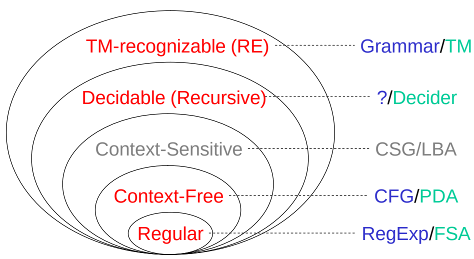
# Unit C2: Overview

- Analyze a realistic example involving a mini language
- Understand CFGs/CFLs
  - Grammars, languages
- Understand how to process CFLs
- Understand the effect of determinism
- Preview Exercise C2 “Context-Free”



Review

# Extended Chomsky Hierarchy




Example grammars?
Essential properties?



Demo

# Graphing Tool

- Basic functions
  - Constant: e.g., **1** (integers) and **2.34** (**.5** must be written as **0.5**)
  - Factorial: only the fixed form of **n!**
  - Logarithm: **log** *OptBase NonConstantExp* (*OptBase*: const)
  - Power: *Exp***^***Exp* where *Exp* is a valid expression. *Exp* other than a constant or **n** requires parentheses.
- Complex expression
  - Complex expressions can be formed by using the operators *****, **/**, **+**, **-**, and parentheses **(** and **)**. ***** and **/** takes precedence over **+** and **-**. The operators are left associative.
- Notes
  - Spaces are ignored internally.
  - **sqrt(n)** must be entered as **n^0.5**.

How to specify?
How to process?

`http://www.tcnj.edu/~komagata/Graphing`



# Group Exercise 1

- How would you implement the module to analyze the input expressions?
  - How to specify?
  - How to process?



# Context-Free Grammar (CFG)

- **G** = $(N, T, R, S)$
  - $N$: finite set of nonterminals [upper case]
  - $T$: finite set of terminals [lower case]
  - $R$: finite set of rules $A \rightarrow \alpha$ where $\alpha$ is a string made up of the elements of N and T
  - $S$: start symbol $\in N$

Formal definition for a CFG for graphing tool?

## Context-Free Language (CFL)

- Derivation (generation) of a string in a CFG, **G** (from the start symbol $S$)
  - $S \rightarrow^* w$ (i.e., zero or more rule application)
  - Also said: "**G** generates $w$" or "**G** accepts $w$"
- $L(\mathbf{G}) = \{w \mid S \rightarrow^* w$, a string made up of the elements of $\mathrm{T}\}$
- CFL = $\{L(\mathbf{G}) \mid \mathbf{G}$ is a CFG$\}$



## Parsing

- Parsing: Process of analyzing how a particular string can be generated by a grammar
- Top-down: Start from the "start symbol"
- Bottom-up: Start from the string
- Hybrid: Combination of both



## Shorthand

- Alternatives: $\mathrm{A} \rightarrow \alpha \mid \beta$
  - $\mathrm{A} \rightarrow \alpha$
  - $\mathrm{A} \rightarrow \beta$
- Optional element: $\mathrm{A} \rightarrow \alpha\ [\beta]$
  - $\mathrm{A} \rightarrow \alpha$
  - $\mathrm{A} \rightarrow \alpha\, \beta$



Demo

## Top-down Parsing

- Example: Recursive-descent parsing
  - Keep the current input and the remaining part of the rule on stack
  - Expand nonterminals
  - Check terminals against the input

- $\mathrm{S} \rightarrow \mathrm{A}$
- $\mathrm{A} \rightarrow \mathrm{a\ A\ b} \mid \mathrm{a\ b}$



Demo

## Bottom-up Parsing

- Example: Shift-reduce parsing
  - If the RHS of a rule matches a part of the input, reduce it to the LHS symbol
  - Otherwise, push the leftmost symbol onto the stack and repeat (shift)

Essential component for processing?

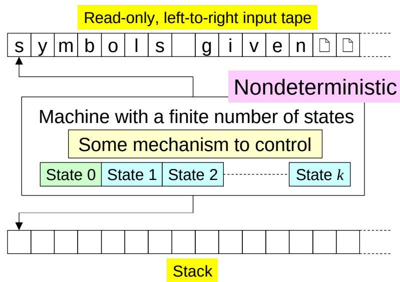


## Main CF Property

- Matching growth to the left and right
  - E.g., $\underbrace{(...(}_{n}\ \underbrace{)...)}_{n}$, $0^n1^n$, $a_1...a_na_n...a_1$
- Characterized by *balanced* rules
  - E.g., $\mathrm{A} \rightarrow (\ \mathrm{A}\ )$, $\mathrm{A} \rightarrow 0\ \mathrm{A}\ 1$, $\mathrm{A} \rightarrow a_i\ \mathrm{A}\ a_i$
- Characterized by the use of stack
  - E.g., pushing '(' and later popping it and matching with ')'

$0^n1^n2^k3^k$, $0^n2^k3^k1^n$, $0^i2^k3^k0^{(n-i)}1^n$

## Push-Down Automata (PDA)






## PDA, Formally

- $\mathbf{M} = (Q, \Sigma, \Gamma, \delta, q_0, F)$
  - $Q$: set of states
  - $\Sigma$: set of input symbols
  - $\Gamma$: set of stack symbols
  - $\delta$: transition function Note: $\varepsilon$ as the empty string

    $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow P(Q \times \Gamma^*)$ power set

    $\Gamma^*$: string of stack symbols
  - $q_0$: initial state $\in Q$
  - $F$: set of final states $\subseteq Q$



## Equivalence of CFGs and PDAs

- Simulate a CFG with a PDA
- Simulate a PDA with a CFG



## Group Exercise 2

- Are there any potential parsing problem the following grammars?

A
- $S \rightarrow A$
- $A \rightarrow a\ A\ b \mid a\ b$
- $A \rightarrow \varepsilon$

B
- $S \rightarrow A$
- $A \rightarrow a\ A\ a$
- $A \rightarrow b$

C
- $S \rightarrow A$
- $A \rightarrow a\ A\ a$
- $A \rightarrow a$



## Ambiguity

- Multiple derivations of a single string
- A complete analysis requires nondeterminism.
- Problems with respect to:
  - Semantics
  - Efficiency



## Deterministic PDA (DPDA)

- PDA where the range of the transition function is a set of singletons (unique state).
  - Equivalently,

    PDA: $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow P(Q \times \Gamma)$

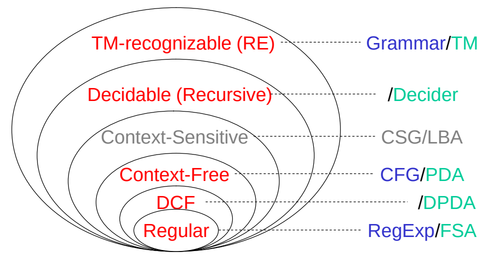
    DPDA: $Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow Q \times \Gamma$
- Sufficiently powerful to characterize programming language core
- Efficient parsing algorithms are known.

## Deterministic CFL (DCFL)

- A subset of CFLs that can be processed by a DPDA
- No easily identified class of grammars
  - Practical issue for specifying a programming language



## Extended Chomsky Hierarchy






## Unit Summary

- Many mini languages can be specified and processed by CFLs and PDAs.
- CFG: A single nonterminal on the LHS (e.g., $A \rightarrow \alpha$)
- CFL: Specified by CFGs
- PDA: Process CFLs
- DPDA/DCFL: Deterministic subset of PDAs/CFLs ~ backbone of programming languages



## Summary Question

- Can you see the “context-free” property in some real-life phenomena? Explain.
- Questions/Comments/Suggestions

Name: ______________________

# Exercise C2, 3/18/05

## Part 1: Review "Context-Free"

The notion of "context-freeness" is extremely important in computer science, due to its tight connection with programming languages. If you understand this class, you understand the backbone of programming languages. So, here are some review questions. It is up to you how to use these. If you understand well by now, you do not need to write up your responses. But recall that passive and active knowledge are different.

Review questions

A. What is the most important property of "context-freeness"?
B. Why the constraint |LHS| = 1 leads to a CFG (cf. unrestricted grammars)?
C. What kind of strings cannot be specified by CFGs?
D. What are the differences between top-down and bottom-up parsing?
E. Why can PDAs capture all of CFLs?
F. Why is the class of DCFLs smaller than that of CFLs?

## Part 2: Graphing Tool (Unit C2 Group Exercise 1)

When we analyze the growth rates of two functions, it would be useful to visualize how fast those functions would grow. The demo program shown in class ([http://www.tcnj.edu/~komagata/Graphing](http://www.tcnj.edu/~komagata/Graphing)) just does this. In this exercise, continue the in-class exercise and give a specification of the language the graphing tool would accept as a CFL. By trying various expressions, you should be able to tell the range of acceptable forms. Although you are encouraged to do this exercise from scratch, if you need something to start with, you can use the following partial information.

- Expression is a series of Terms connected with + or –
    - Exp → Term
    - Exp → Term + Exp
    - Exp → Term – Exp
- Term is a series of Factors connected with * or /
- Factor is Const, Factorial, Log, or Power
- Const
- Factorial → "n!"
- Log
    - Log → "logn" | "log" Const "n" | "log(" Exp ")" | "log" Constant "(" Exp ")"
    - Note: "log" Const Const is illegal
    - Note: not including "log(" Constant ")(" Exp ")"
- Power
    - Power → ("n" | "(" Exp ")" | Const) "^" ("n" | "(" Exp ")" | Const)

## Part 3: Foxtrot

CFGs are quite useful for various things. In my Discrete Math courses, I used examples such as robot navigation, DNA analysis, dance choreography (sort of), etc. Here is another example from ballroom dancing. A collection of steps of "Foxtrot" can be described by the following CFG (only the rewrite rules are shown; other information can be derived from them):

| | | | |
|---|---|---|---|
| Foxtrot | → | Basic | |
| Basic | → | Basic Basic | |
| Basic | → | **featherFinish** WeaveSeq Turn | |
| Basic | → | **promenadePosition naturalWeave** Turn | |
| WeaveSeq | → | ε | (empty string, i.e., no steps) |
| WeaveSeq | → | **weave** WeaveSeq | |
| Turn | → | **threeStep heelTurn** | |

Here, Foxtrot (start symbol), Basic, WeaveSeq, and Turn (beginning with an uppercase letter) are nonterminals, and the others are terminals. We will and you may abbreviate nonterminals/terminals using the first letter (case sensitive), e.g., W for WeaveSeq and **w** for **weave**.

A. Would the following step sequence be acceptable according to the above description? If yes, show how such a sequence can be generated. If no, explain.

   **f t h f w t h p n t h f w w w t h**

B. Discuss the advantages/disadvantages of using (a) top-down parsing and (b) bottom-up parsing for this grammar. Note that you must contrast top-down vs. bottom-up approaches in general and should *not* be limited to some particular top-down or bottom-up mechanisms (i.e., do not discuss recursive descent, LL(1), LR(k), etc.).

   Note: You must clearly state some contrastive feature(s).

C. Would it be possible to generate exactly the same language (the step sequences generated by the above-mentioned CFG) using a regular grammar? Explain. In addition, argue whether this language is a context-free or regular [your argument should be very brief].

   Note: A regular grammar is like CFGs except that the RHS can have only one nonterminal at the beginning or at the end. For example, a regular grammar can have rules such as "A → a b C" and "A → A b c", but *not* "A → a B c" or "A → a B C" (uppercase symbols are nonterminals). Regular grammars is equivalent to regular expressions and finite-state automata.

   Hint: Examine several example step sequences and figure out the pattern of generated strings. Is the pattern similar to that of parenthesis matching, which requires a CFG?

// End

# Ex C2 Part 1 “CF”

A. What is the most important property of “context-freeness”?
B. Why the constraint |LHS| = 1 leads to a CFG (cf. unrestricted grammars)?
C. What kind of strings cannot be specified by CFGs?
D. What are the differences between top-down and bottom-up parsing?
E. Why can PDAs capture all of CFLs?
F. Why is the class of DCFLs smaller than that of CFLs?



Review

# Languages/Grammars/Automata

- Language: Set of strings ~ problems
- Grammar: Convenient way of specifying languages
  - E.g., $L(G)$ for a CFG $G$
- Automata: Abstract model of processing languages
  - E.g., $L(G)$, where $G$ is a CFG, needs a PDA
- Chomsky hierarchy: about languages



# Ex C2 Part 2 Graphing Tool

- Expression is a series of Terms connected with + or -
  - Exp → Term  Exp → Term + Exp  Exp → Term - Exp
  - Note: Exp → ε
- Term is a series of Factors connected with * or /
  - Term → Factor  Term → Factor * Term  Term → Factor / Term
- Factor is Const, Factorial, Log, or Power
  - Factor → Const | Factorial | Log | Power
- Const
  - Const → Digits  Const → Digits “.” Digits  Digits → Digit
  - Digits → Digit Digits  Digit → 0 | 1 | .... | 9
- Factorial → “n!”
- Log
  - Log → “logn” | “log” Const “n” | “log(” Exp “)” | “log” Constant “(“ Exp “)”
  - Note: “log” Const Const is illegal
  - Note: not including “log(” Constant “)(” Exp “)”
- Power
  - Power → (“n” | “(” Exp “)” | Const) [ “^” (“n” | “(” Exp “)” | Const) ]



# Ex C2 Part 3 Foxtrot Question C

- Possible to represent the same language with a regular grammar?

| | | |
|---|---|---|
| Foxtrot | → | Basic |
| Basic | → | **promenadePosition naturalWeave threeStep heelTurn** Basic |
| Basic | → | **featherFinish** WeaveNext |
| WeaveNext | → | **weave** WeaveNext |
| WeaveNext | → | **threeStep heelTurn** [ Basic ] |



Demo

# Top-down Parsing

- Example: Recursive-descent parsing
  - Keep the current input and the remaining part of the rule on stack
  - Expand nonterminals
  - Check terminals against the input

- S → A
- A → a A b | a b



Demo

# Bottom-up Parsing

- Example: Shift-reduce parsing
  - If the RHS of a rule matches a part of the input, reduce it to the LHS symbol
  - Otherwise, push the leftmost symbol onto the stack and repeat (shift)

`http://www.tcnj.edu/~komagata/ShiftReduce/`

## Main CF Property

- Matching growth to the left and right
  - E.g., $\underbrace{(...(}_{n}\underbrace{)...)}_{n}$, $0^n1^n$, $a_1...a_na_n...a_1$
- Characterized by *balanced* rules
  - E.g., $A \to ( A )$, $A \to 0 A 1$, $A \to a_i A a_i$
- Characterized by the use of stack
  - E.g., pushing '(' and later popping it and matching with ')'

$0^n1^n2^k3^k$, $0^n2^k3^k1^n$, $0^i2^k3^k0^{(n-i)}1^n$



## Group Exercise 1 (CF)

- Which of the following languages is context-free? Why?

1. $0^n1^n2^k3^k$
2. $0^n2^k3^k1^n$
3. $0^i2^k3^k0^{(n-i)}1^n$

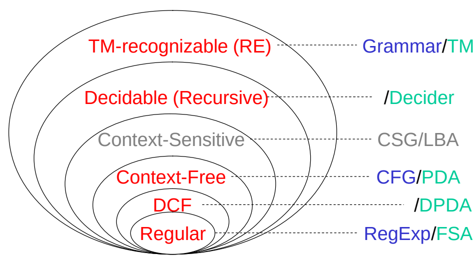


## Unit C3: Overview

- Review examples of regular languages
  - Review grammars/automata for regular languages
- Understand the effects of nondeterminism for different automata
- Preview Exercise C3 "Regular"



## Extended Chomsky Hierarchy






Initial survey problem

## Vending Machine

- Imagine a vending machine that accepts nickels, dimes, and quarters and sells Vodka by a paper cup for 35 cents. Draw a diagram of a finite-state machine/automaton that would represent the state of the vending machine. Make additional assumptions if necessary.



## Password Screening

- Requirement 1
  - Your password must be non-TM-recognizable.
- Requirement 2
  - Your password must be a palindrome.
- Requirement 3
  - Your password must be at least 6 characters long, contain at least one uppercase letter, *and* at least one symbol *or* a numeral.

# Java Comments

- End-of-line comment: `// ...` (until EOL)
- Traditional comment: `/* ... */`
  - Note: No nesting allowed



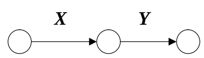
# Binary Arithmetic

- Do we need a TM?
- Do we need a PDA?

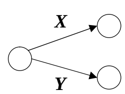


# FA Components

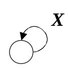
- Sequence




- Alternative



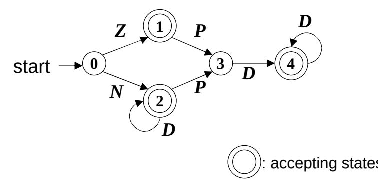

- Repetition (zero or more times)






# Finite-State Automata (FSA)

- A machine that consists of the three components shown on the previous slide
- Example



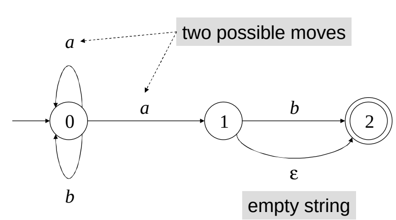



# Deterministic FA (DFA)

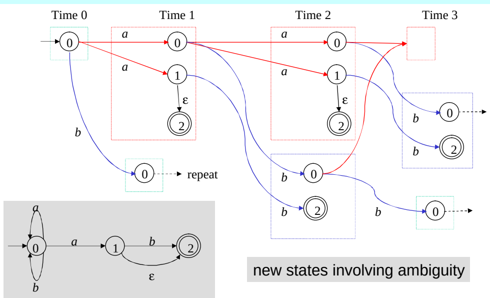
- $\mathbf{M} = (Q, \Sigma, \delta, q_0, F)$
  - $Q$: set of states
  - $\Sigma$: set of input symbols
  - $\delta$: transition function
    $Q \times \Sigma \rightarrow Q$
  - $q_0$: initial state $\in Q$
  - $F$: set of final states $\subseteq Q$



# Nondeterministic Example



# Tracing Example 1






# Observation

- Due to nondeterminism and ε-transitions, the result of a move can be a combination of possible next states.
- For a set of states $Q$, such a combination is a member of the power set of $Q$, i.e., $P(Q)$.
  - E.g., $P(\{1, 2\}) = \{\varnothing, \{1\}, \{2\}, \{1, 2\}\}$
- Naturally, any DFA is an NFA.



# Nondeterministic FA (NFA)

- $\mathbf{M} = (Q, \Sigma, \delta, q_0, F)$
  - $Q$: set of states
  - $\Sigma$: set of input symbols
  - $\delta$: transition function
    $Q \times (\Sigma \cup \{\varepsilon\}) \rightarrow P(Q)$ cf. transition relation
  - $q_0$: initial state $\in Q$
  - $F$: set of final states $\subseteq Q$



# Nondeterminism

- Possibility of multiple transitions from a state on a single input
- Instead of a single state, considering a set of states
  - $S$ as the set of all states
  - $P(S)$ as the power set of $S$
  - The next state $\in P(S)$
- Power set ~ nondeterminism

exponential growth



# Effects of Nondeterminism

- TMs: No effects on power (i.e., the set of languages that can be recognized)
- PDAs: Affects the power (i.e., PDAs ≠ DPDAs)
- FSAs: No effects on the power (i.e., the set of languages that can be recognized)
- Efficiency/speed: exponential slow down



# Regular Grammar

- $\mathbf{G} = (N, T, R, S)$
  - $N$: finite set of nonterminals [upper case]
  - $T$: finite set of terminals [lower case]
  - $R$: finite set of rules $A \rightarrow \alpha\, B$ or $A \rightarrow B\, \alpha$ where
    - $\alpha$ is a string made up of the elements of $T$
    - $B \in N$
  - $S$: start symbol $\in N$
- Regular languages = $\{\mathbf{L}(\mathbf{G}) \mid \mathbf{G}$ is a regular grammar$\}$

## Regular Expression (RegExp)

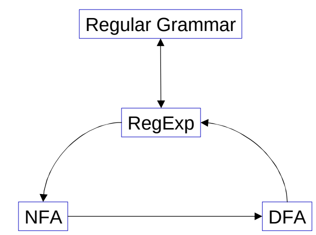
- Practical alternative to regular grammar
- Expression components (with sets $X$, $Y$)
  - Sequence: $\boldsymbol{X\ Y}$  Notation: $X^n = \underbrace{X \ldots X}_{n}$
  - Alternative: $\boldsymbol{X} \mid \boldsymbol{Y}$
  - Repetition (zero or more times): $\boldsymbol{X^*}$ Kleene closure

A single regular expression represents a *set*.
Convention: Singleton $\{\mathbf{a}\}$ abbreviated as **a**

- Equivalence of RegExp and regular grammars



## Equivalence






## Main *Regular* Property

- Language: Uncoordinated repetition
  - E.g., $0^m1^n$, “a very very .... long noun phrase”
- Grammar: Characterized by recursion *at edge*
  - E.g., $\mathrm{A} \rightarrow a\ \mathrm{A}$, $\mathrm{A} \rightarrow \mathrm{A}\ a$
  - Equivalently, Kleene closure, $X^*$
- Automaton: Characterized by finite states
  - E.g., looping represented by a cycle



## More Examples

- FP numbers
- URL
- Spanish spelling (cf. English spelling)
- Noun phrase in English (except for CF components)
- Musical code sequence (?)
- Your daily tasks



## Unit Summary

- Main property of regular languages: *uncoordinated* repetition, cf. CF
- RegExp is a practical way of specifying a broad range of simple languages.
- FA is a simple and fast model for processing regular languages.



## Summary Question

- How far have you been thinking Exercise C5 (Comprehensive Exercise)? Explain. If not yet, what is your current thought?
- Questions/Comments/Suggestions

Name: ______________________

# Exercise C3, 3/22/05

## Part 1: Language Identification

Regular expressions are one of the most commonly used way to specify regular languages. Since most modern computing environment support the use of regular expression, it would be good to practice specifying a regular language using a regular expression.

As an example, let us observe a very crude way to identify Japanese names from a list of names (http://www.tcnj.edu/~komagata/csc460/05s/misc/names.txt). The first approximation is to pick up the basic Japanese syllabic structure, which is a repetition of consonant-vowel sequence. A regular expression $(CV)^+$ captures the pattern, where $V$ = {**a**, **e**, **i**, **o**, **u**} and $C$ = *LowerCaseLetters* – *V*. Using the Unix **egrep** utility, we can see how well this pattern picks up Japanese names.

```
% egrep -ni '^([bcdfghjklmnpqrstvwxyz][aeiou])+$'
   ~komagata/www/csc460/05s/misc/names.txt
83:Duda
142:Ho
150:Hu
166:Kane
177:Ko
197:Levi
198:Levine
199:Li
243:Pi
245:Pulito
250:Rice
259:Sami
299:Su
324:Vita
325:Vu
345:Wu
346:Xu
348:Ye
350:Yi
351:Yi
354:Yu
```

Since most names are not really Japanese (although you may not be able to tell), this pattern is not really good. But the accuracy can be improved by fine tuning the pattern.

**Task**: Pick one human language and create a single regular expression that would pick up the names in that language to some extent. Include the regular expression and a test run on the name list. Human languages are full of exceptions. So, limit your time and satisfy yourself at some point.

Note: If you prefer, you can use an alternative programming language/environment that supports regular expressions, e.g., perl, PHP.

## Part 2: Distinguishing Regular and Non-Regular Languages

FSAs are extremely simple and fast automata. So, if your problem (set, language) is regular, you should definitely use an FSA, rather than a PDA or a TM as the abstract model for developing a program. Then, the question would be to analyze whether the problem is regular.

**Task 1**: Consider the following problem: Liisa realized that for any day, she receives $m$ important (i.e., non-junk) e-mail messages followed by $n$ junk messages, where $m + n = 2k$ for some integer $k \geq 1$. First, represent the problem as a set. Then, identify whether the problem, interpreted as a language, is regular or non-regular. Justify your answer at least to the level discussed in class.

**Task 2**: Consider the following problem: Mikko realized that whenever he wins the card game $m$ consecutive times, he loses $n$ consecutive times after that, where $n = 2m + 1$. First, represent the problem as a set. Then, identify whether the problem, interpreted as a language, is regular or non-regular. Justify your answer at least to the level discussed in class.

// End

## Ex C3 Part 2

- **Task 1**: Liisa realized that for any day, she receives $m$ important (i.e., non-junk) e-mail messages followed by $n$ junk messages, where $m + n = 2k$ for some integer $k \geq 1$.
- **Task 2**: Mikko realized that whenever he wins the card game $m$ consecutive times, he loses $n$ consecutive times after that, where $n = 2m + 1$.



## Unit C4: Overview

- Distinguish regular and non-regular languages
  - Pumping Lemma (for regular languages)
- Understand and use the properties of regular languages
  - Closure properties
  - Equivalence of regular languages
- Preview Exercise C4 “Regular Practice”



Preview

## Regular?

- Top-down approach to paper writing
- Finite language
- $0^n1^n$
- $\{0^m1^n \mid m, n \geq 0\}$
- $\{0^m1^n \mid m \neq n\}$
- $\{w\$w^R \mid w^R$ is the reverse of $w \in \{0, 1\}^*\}$
- $\{ww^R \mid w^R$ is the reverse of $w \in \{0, 1\}^*\}$
- $\{ww \mid w \in \{0, 1\}^*\}$
- $\{ww^Rw \mid w^R$ is the reverse of $w \in \{0, 1\}^*\}$



## Analysis of “Regular”ness

- To Show that $L$ is “regular”
  - Proof by existence: Give a regular grammar, RegExp, or FSA
- To Show that $L$ is *not* “regular”
  - Need to prove that no regular grammar (or RegExp or FA) can generate the language
  - Demonstrate that some property of regular languages cannot hold



## Main Property of Regular Languages

- *Uncoordinated* repetition
- Inputs longer than the number of FA states $\Rightarrow$ Some state(s) must be repeated
- Any number of that repetition must result in an acceptable input.

cf. pumping gas for free



## Pumping Lemma (for regular languages)

To show that a language is *not* regular

- For any <u>infinite</u> regular language $L$,
- there exists a positive integer $n_0$ such that
- for any $w \in L$ such that $|w| \geq n_0$,
- there exists a decomposition $w = xyz$ where $|xy| \leq n_0$ and $|y| \geq 1$ such that
- for any $i \geq 0$,
- $xy^iz \in L$

## Game-Theoretic Interpretation of FOL

cf. logic-structure connection

- $\forall x \exists y$ ($x$ kicks $y$)
  - Falsifier (tries to crack $\forall$): Choose $x$
  - Verifier (tries to support $\exists$): Choose $y$, based on $x$
  - Check whether "$x$ kicks $y$" is true
- $\exists x \forall y$ ($x$ kicks $y$)
  - Verifier: Choose $x$
  - Falsifier: Choose $y$, based on $x$
  - Check whether "$x$ kicks $y$" is true

To satisfy: Play a verifier
To reject: Play a falsifier



## Group Exercise 1

Satisfy/reject the following

A. Everyone in this class is taking at least one upper-level CS course.

B. Some cat loves every dog at some point.

C. Some cat loves every dog at any point.

Procedure:
- Take the role of either the verifier or falsifier
- To satisfy, the verifier must win.
- To reject, the falsifier must win.



## Main Property of Regular Languages

Revisited

- *Uncoordinated* repetition
- Inputs longer than the number of FA states $\Rightarrow$ Some state(s) must be repeated
- Any number of that repetition must result in an acceptable input.



## Logic Game for Pumping Lemma

Between a verifier (for $\exists$) and a falsifier (for $\forall$)

- Falsifier: Choose an infinite language $L$
- Verifier: Choose a positive integer $n_0$ such that
- Falsifier: Choose $w \in L$ such that $|w| \geq n_0$,
- Verifier: Choose a decomposition $w = xyz$ where $|xy| \leq n_0$ and $|y| \geq 1$ such that
- Falsifier: Choose $i \geq 0$,
- Check whether $xy^iz \in L$

Meaning of $|xy| \leq n_0$

To satisfy: Play a verifier (wrong)
To reject: Play a falsifier (correct)



## Misusing Pumping Lemma

Play a verifier (Note: *not* proving that $L$ is regular; *not useful*)

- To show that $0^m1^n$ has "regular"ness
- Choose $n_0 = 1$
- Anticipate any $0^m1^n \in L$ such that $m + n \geq 1$
- Choose a decomposition $w = xyz$:
  - Case 1 ($\mathrm{m} = 0$): $1^m\ 1^c\ 1^d$ ($c \geq 1$)
  - Case 2 ($\mathrm{m} \neq 0$): $0^a\ 0^b\ 1^n$ ($b \geq 1$)
- Anticipate any $i \geq 0$
- $0^a0^{b\times i}1^n \in L$, $0^m1^{c\times i}1^d \in L$



## Using Pumping Lemma

Play a falsifier: This is how this lemma is used.

- To show that $0^n1^n$ is *not* regular
- Anticipate any $n_0 \geq 1$
- Choose $0^n1^n \in L$ such that $2n \geq n_0$
- Anticipate any decomposition $w = xyz$:
  - Case 1: $0^a\ 0^b\ 1^n$ ($b \geq 1$)
  - Case 2: $0^n\ 1^c\ 1^d$ ($c \geq 1$)
  - Case 3: $0^a\ (0^b1^c)\ 1^d$ ($\mathrm{b} + \mathrm{c} \geq 1$)
- Choose $i = 2$
- $0^a0^{2b}1^n \notin L$, $0^n1^{2c}1^d \notin L$, and $0^a(0^b1^c)^21^d \notin L$

## Regular?

- Top-down approach to paper writing
- Finite language
- $0^n 1^n$
- $\{0^m 1^n \mid m, n \geq 0\}$
- $\{0^m 1^n \mid m \neq n\}$
- $\{w\$w^R \mid w^R$ is the reverse of $w \in \{0, 1\}^*\}$
- $\{ww^R \mid w^R$ is the reverse of $w \in \{0, 1\}^*\}$
- $\{ww \mid w \in \{0, 1\}^*\}$
- $\{ww^R w \mid w^R$ is the reverse of $w \in \{0, 1\}^*\}$



## Group Exercise 2

Satisfy/reject the following

A. $\{w\$w^R \mid w^R$ is the reverse of $w \in \{0, 1\}^*\}$

B. $\{ww^R \mid w^R$ is the reverse of $w \in \{0, 1\}^*\}$

Procedure:
- Take the role of either the verifier or falsifier
- To satisfy, the verifier must win.
- To reject, the falsifier must win.



## Closure

Closed under
- Concatenation
- Union
- Kleene closure
- Complement
- Intersection



## Equivalence of Regular Languages

- Equivalence of States
  - Equivalent behavior for all strings with respect to acceptance
  - To find out: compare every pair of states against all strings (systematically)
- Minimization of DFAs
  - Merge equivalent states $\Rightarrow$ partition
- Equivalence of regular languages
  - Merge two DFAs and test equivalence of the two start states



## Unit Summary

- Regular vs. non-regular
  - To show regular: Construct a regular grammar, RegExp, or a FSA
  - To show not regular: Use the Pumping Lemma
- Properties
  - Closed under most common operations
  - Unique minimization possible
  - Easy to manipulate regular languages and their specification/processing mechanisms



## Summary Question

- The Pumping Lemma is tricky. You must have questions. What are they?
- List other questions as well, if you have.

# Unit C4 Supplement, 3/25/05

## Pumping Lemma (for regular languages)

Since I was not able to explain this well in class, I will add a few notes. In addition, the slides are in an abbreviated form. So, I will try to give more details here.

First, the Pumping Lemma says the following (informally). Given an infinite regular language, we can always find a (sufficiently long) string ($w$) that has a substring ($y$) contributing to (uncoordinated) repetition, and that the substring can be pumped zero or more times ($y^*$). Although we don't prove this lemma, we can intuitively tell this is true.

As discussed in class, this lemma is used only for showing that certain languages are not regular. To do this, we take the role of a falsifier and face a competing verifier.

- As a falsifier, choose an infinite (supposedly) regular language $L = 0^n1^n$ (shorthand for $\{0^n1^n \mid n \geq 0\}$). If this was indeed regular, the falsifier could never win.
- The opponent, a verifier, chooses a positive integer $n_0$, say, 4. This could be any other positive integer, though.
- We choose $w = 0^21^2 \in L$ , which satisfies $|0^21^2| = 4 \geq n_0 = 4$.
- The opponent can decompose $w$. The opponent only needs to choose the decomposition most damaging to us. However, since we don't know what it is, we consider all the possibilities of decomposing $w = xyz$, where $|xy| \leq n_0 = 4$ and $|y| \geq 1$
    - Case 1 ($y$ stays within $0^2$)
        - Subcase A: $x = 0$, $y = 0$, z = 11
        - Subcase B: $x = \varepsilon$, $y = 00$, z = 11
    - Case 2 ($y$ stays within $1^2$)
        - Subcase A: $x = 00$, $y = 1$, z = 1
        - Subcase B: $x = 00$, $y = 11$, z = $\varepsilon$
    - Case 3 ($y$ crosses $0^21^2$)
        - Subcase A: $x = 0$, $y = 01$, z = 1
        - Subcase B: $x = \varepsilon$, $y = 001$, z = $\varepsilon$
        - Subcase C: $x = 0$, $y = 011$, z = $\varepsilon$
        - Subcase D: $x = \varepsilon$, $y = 0011$, z = $\varepsilon$
    - Note: Depending on the choice made in the previous steps, there may be more cases/subcases. But the idea is analogous.
- We choose $i = 2 \geq 0$. Then, for Cases 1 and 2, the balance between 0's and 1's will be destroyed. For Case 3, the 0-to-1 ordering will be destroyed. In any case, we defeated the opponent. This means that we have rejected the Pumping Lemma. However, since the lemma holds for any infinite regular languages, we conclude that the language is not regular. [Strictly speaking, this is a proof by contradiction.]

The problem $w\$w^R$ can be analyzed in an analogous way. In particular, you might consider different cases: whether or not $y$ includes \$.

For $ww^R$, a similar technique may not work well. Recall the class discussion, where you are forced to pump the middle section of, say, 00$\underline{11}$00. You *can* pump and will still get strings that are in the language. To see how to deal with this problem, you may want to re-visit $0^n1^n$ and re-examine the choices made by the players. There are ways for the falsifier to defeat the verifier more easily. For example, what would happen if you consider $w$ twice as long as $n_0$ (instead of $|w| = n_0$ as in the earlier example)? If it leads to a simpler strategy, what makes it happen? Such an idea may be useful for showing that $ww^R$ is not regular.

If you have questions, try to post on the discussion board, although you can always send them to me.

// End

Name: ______________________

# Exercise C4, 3/25/05

## Part 1: Review: First-Order Logic (FOL)

The Pumping Lemma involves a complicated interaction of first-order quantifiers ($\forall$ and $\exists$). In order to understand how to use it, we need to know the basics of FOL.

**Task** (optional): Review FOL, if necessary.

## Part 2: Game-Theoretic Interpretation of FOL

One way to interpret complex first-order statements is to use the game-theoretic interpretation.

**Task 1**: Write up Unit C4 Group Exercise 1.

**Task 2** (optional): '$O$' (asymptotic upper bound) is defined as follows: $f(n) \in O(g(n))$ if there are constants $c > 0$ and $n_0 \geq 1$ such that $f(n) \leq c\, g(n)$ for every integer $n \geq n_0$. Use the game-theoretic interpretation to analyze this.

**Task 3** (optional): The limit $c$ of a function $f(x)$ as $x$ approaches $a$, $\lim_{x \to a} f(x) = c$, is defined as follows: For every real number $\varepsilon > 0$, there exists a real number $\delta > 0$ such that $|f(x) - c| < \varepsilon$ whenever $0 < |x - a| < \delta$. Use the game-theoretic interpretation to analyze this.

## Part 3: Distinguishing Regular vs. Non-Regular Languages

Using the Pumping Lemma, we can prove that a certain language is *not* regular in a precise manner. For such languages, we will need to use a more powerful mechanism, e.g., PDA, TM.

**Task 1**: Write up Unit C4 Group Exercise 2.

**Task 2**: Redo the two problems (tasks) in Exercise C3 Part 2 using the tools discussed in Unit C4. Note that you will need to use different approaches to show that a language is regular or non-regular.

Survey: Time spent between classes: ______________

Unit C4 slides for the in-class group exercises and take-home exercies.

## Game-Theoretic Interpretation of FOL

cf. logic-structure connection

- $\forall x \exists y$ ($x$ kicks $y$)
  - Falsifier (tries to crack $\forall$): Choose $x$
  - Verifier (tries to support $\exists$): Choose $y$, based on $x$
  - Check whether "$x$ kicks $y$" is true
- $\exists x \forall y$ ($x$ kicks $y$)
  - Verifier: Choose $x$
  - Falsifier: Choose $y$, based on $x$
  - Check whether "$x$ kicks $y$" is true

To satisfy: Play a verifier
To reject: Play a falsifier

CSC460 C4 7

## Pumping Lemma (for regular languages)

To show that a language is *not* regular

- For any <u>infinite</u> regular language $L$,
- there exists a positive integer $n_0$ such that
- for any $w \in L$ such that $|w| \geq n_0$,
- there exists a decomposition $w = xyz$ where $|xy| \leq n_0$ and $|y| \geq 1$ such that
- for any $i \geq 0$,
- $xy^iz \in L$

CSC460 C4 6

## Logic Game for Pumping Lemma

Between a verifier (for $\exists$) and a falsifier (for $\forall$)

- Falsifier: Choose an infinite language $L$
- Verifier: Choose a positive integer $n_0$ such that
- Falsifier: Choose $w \in L$ such that $|w| \geq n_0$,
- Verifier: Choose a decomposition $w = xyz$ where $|xy| \leq n_0$ and $|y| \geq 1$ such that
- Falsifier: Choose $i \geq 0$,
- Check whether $xy^iz \in L$

Meaning of $|xy| \leq n_0$

To satisfy: Play a verifier (wrong)
To reject: Play a falsifier (correct)

CSC460 C4 10

## Using Pumping Lemma

Play a falsifier: This is how this lemma is used.

- To show that $0^n1^n$ is *not* regular
- Anticipate any $n_0 \geq 1$
- Choose $0^n1^n \in L$ such that $2n \geq n_0$
- Anticipate any decomposition $w = xyz$:
  - Case 1: $0^a$ $0^b$ $1^n$ ($b \geq 1$)
  - Case 2: $0^n$ $1^c$ $1^d$ ($c \geq 1$)
  - Case 3: $0^a$ $(0^b1^c)$ $1^d$ (b + c $\geq$ 1)
- Choose $i = 2$
- $0^a0^{2b}1^n \notin L$, $0^n1^{2c}1^d \notin L$, and $0^a(0^b1^c)^21^d \notin L$

CSC460 C4 12

// End

## Pumping Lemma (for regular languages)

To show that a language is *not* regular

- For any infinite regular language $L$,
- there exists a positive integer $n_0$ such that
- for any $w \in L$ such that $|w| \geq n_0$,
- there exists a decomposition $w = xyz$ where $|xy| \leq n_0$ and $|y| \geq 1$ such that
- for any $i \geq 0$,
- $xy^iz \in L$



## Using Pumping Lemma

Play a falsifier: This is how this lemma is used.

- To show that $0^n1^n$ is *not* regular
- Anticipate any $n_0 \geq 1$
- Choose $0^n1^n \in L$ such that $2n \geq n_0$
- Anticipate any decomposition $w = xyz$:
  - Case 1: $0^a$ $0^b$ $1^n$ $(b \geq 1)$
  - Case 2: $0^n$ $1^c$ $1^d$ $(c \geq 1)$
  - Case 3: $0^a$ $(0^b1^c)$ $1^d$ (b + c ≥ 1)
- Choose $i = 2$
- $0^a0^{2b}1^n \notin L$, $0^n1^{2c}1^d \notin L$, and $0^a(0^b1^c)^21^d \notin L$



## $w\$w^R$ is not regular

Play a falsifier

- To show that $w\$w^R$ is *not* regular
- Anticipate any $n_0 \geq 1$
- Choose $w\$w^R \in L$ such that $|w\$w^R| \geq n_0$
- Anticipate any decomposition $w\$w^R = xyz$ where $|xy| \leq n_0$ and $|y| \geq 1$:
  - Case 1 ($y$ includes \$): $w_1$ $(w_2\$w_2^R)$ $w_1^R$ $[|w_2| \geq 0]$
  - Case 2 ($y$ does not include \$): $w_1$ $w_2$ $(w_3\$w^R)$
- Choose $i = 2$
- $w_1(w_2\$w_2^R)^2w_1^R \notin L$, $w_1w_2^2w_3\$w^R \notin L$



## $ww^R$ is not regular

Play a falsifier

- To show that $ww^R$ is *not* regular
- Anticipate any $n_0 \geq 1$
- Choose $ww^R \in L$ such that $ww^R = 0^k1uu^R10^k$ where $k > n_0$
- Anticipate any decomposition $ww^R = xyz$ where $|xy| \leq n_0$ and $|y| \geq 1$:
  - $0^p$ $0^q$ $(0^r1uu^R10^k)$ where $p + q + r = k$ and $q \geq 1$
- Choose $i = 2$
- $0^p(0^q)^20^r1uu^R10^k \notin L$



## Unit C5: Overview

- Distinguish CF and non-CF languages
  - Pumping Lemma (for CF languages)
- Analyze the effects of combining subproblems (regular or CF)
  - Closure properties
- Wrap up Module C



## Context-Free? [general]

- HTML/XML
- Processing recursive function calls
- Depth-first tree traverse
- Overhauling (repairing a physical device)
- Sample Problem #20 “respectively”
- Dishwashing
- Teaching a course

## Context-Free? [formal]

- $0^n1^n2^n$
- $0^m1^n2^m2^n$
- $\{ww^Rw \mid w^R$ is the reverse of $w \in \{0, 1\}^*\}$
- $\{ww \mid w \in \{0, 1\}^*\}$



## Analysis of CF-ness

- To Show that $L$ is CF
  - Proof by existence: Give a CFG or PDA
- To Show that $L$ is *not* CF
  - Need to prove that no CFG (or PDA) can generate the language
  - Demonstrate that some property of CFLs cannot hold



## Main Property of CFLs

- *Parenthesis matching*
- Use of a single stack
- Arbitrary nesting is possible
  - $0^n1^n$, $0^n1^{n+m}$, $0^n1^{2n}$
  - $0^n2^m1^n$, $0^n2^m3^m1^n$
  - $0^k2^m0^{(n-k)}1^n$, $0^k2^m3^m0^{(n-k)}1^n$

cf. pumping air into the lung



## Pumping Lemma (for CFLs)

To show that a language is *not* CF

- For any CFL $L$,
- there exists a positive integer $n_0$ such that
- for any $w \in L$ such that $|w| \geq n_0$,
- there exists a decomposition $w = abcde$ where $|bcd| \leq n_0$ and $|bd| \geq 1$ such that
- for any $i \geq 0$,
- $ab^icd^ie \in L$



## $0^n1^n2^n$ is *not* CF

Play a falsifier

- To show that $0^n1^n2^n$ is *not* CF
- Anticipate any $n_0 \geq 1$
- Choose $0^n1^n2^n \in L$ such that $|0^n1^n2^n| \geq 3n_0$
- Anticipate any decomposition $0^n1^n2^n = a\underline{b}c\underline{d}e$ where $|\underline{b}c\underline{d}| \leq n_0$ and $|\underline{bd}| \geq 1$:
  - Case 1: $\underline{b}c\underline{d}$ spans part of $0^n1^n$
  - Case 2: $\underline{b}c\underline{d}$ spans part of $1^n2^n$
- Choose $i = 2$
- At least one of 0 and 1 is pumped: i.e., $0^{n+1}1^n2^n$ or $0^n1^{n+1}2^n$ or $0^{n+1}1^{n+1}2^n$



## Group Exercise 1

Context-free? Justify.

A. $\{0^i1^j2^k \mid i = j + k\}$

B. $\{0^i1^j2^k \mid i < j < k\}$

## Combining Subproblems

- Compilers
  - Lexer: Regular
  - Parser: Context-free
  - Semantic analysis: Decidable (ignore the context-sensitive level)
  - Compiler as a whole: ?
- Bird flocking
  - Each bird (per frame): Regular
  - Entire simulation: ?



## Dealing with Subproblems

- Serial connection ~ Concatenation
- Parallel/alternative connection ~ Union
- Repetition ~ Kleene closure
- Simultaneous requirements ~ Intersection
- Negation ~ Complement



## Closure

- A class $C$ is Closed under binary operation $op$.
  - The result of computing $C_1 \; op \; C_2$, where $C_1, C_2 \in C$, is still in $C$.
- A class $C$ is Closed under unary operation $op$.
  - The result of computing $op \; C_1$, where $C_1 \in C$, is still in $C$.



## Closure: Regular

- Concatenation
- Union
- Kleene closure
- Complement
- Intersection



## Closure: Context-Free

- Concatenation
- Union
- Kleene closure
- Intersection
- Complement

Group Exercise 2



## Group Exercise 2

CFLs closed under the following operations?

A. Intersection
B. Complement

## Intersection of CFLs

- $L_1 = 0^n 1^n 2^i \in$ CFL
- $L_2 = 0^i 1^n 2^n \in$ CFL
- $L_1 \cap L_2 = 0^n 1^n 2^n \notin$ CFL

Now, is CFLs closed under complement?

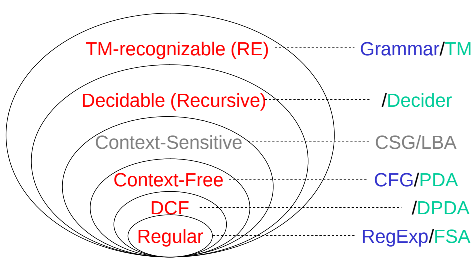


## Unit Summary

- CF vs. non-CF
  - To show CF: Give a CFG or PDA
  - To show not CF: Use the Pumping Lemma
- Closure: Regular languages
  - Closed under concatenation, union, Kleene closure, complement, and intersection
- Closure: CFLs
  - Closed under concatenation, union, and Kleene closure
  - Not closed under complement or intersection



## Extended Chomsky Hierarchy






## Examples

- Pumping
  - Gas (automobile)
  - Air (human lung)
- Computer environment
  - Occurrence of arbitrary events
  - Modularity, recursion, interrupt, etc.
- Baby (newborn)
  - Limb movement
  - Remember things after sleep?



## Module C Summary

- Chomsky hierarchy and properties
  - Regular languages: mostly closed and extremely fast to process
  - CFLs: partially closed and potential processing drawbacks
    - E.g., compilers only deal with DCFLs
  - Beyond: *more* closed but easily undecidable
- Distinguishing classes
  - Constructive vs. Pumping Lemma

# Unit C5 Supplement, 3/29/05

## **Slide 4**: $ww^R$ is not regular

After looking at the slide again, I realized that the slide is correct. The key is to choose $ww^R = 0^k1uu^R10^k$ where $k > n_0$. This way, $xy$ falls on the first $0^k$. By pumping $y$ (e.g., $y^2$), we introduce additional 0's, which will conflict with the 1 preceding the last $0^k$.

## **Slide 12: Group Exercise 1 Problem A.** $\{0^i1^j2^k \mid i = j + k\}$

The question was as follows. By choosing *w* in a way $bcd \leq i$, the falsifier could show that any decomposition of *w* would destroy the context-freeness. Wouldn't it suggest that the language is not context-free?

The Pumping Lemma for CFLs is slightly different from the version for regular languages in that the span of *bcd* can move around more flexibly due to the lack of constraints on *a*. So, for this case, the verifier will be able to slide *bcd* so that *b* and *d* would fall on parts of 0 and 1, respectively, to allow arbitrary pumping.

// End

Name: ______________________

# Exercise C5 (Module C Comprehensive Exercise), 3/29/05

Note: This exercise is announced in advance so that we can think about it through this module. The due date of this exercise is the date of Module C Evaluation Workshop.

## Part 1: Identifying the Simplest Mechanism Along the Extended Chomsky Hierarchy

One of the main goals of Modules C is to be able to identify the minimal mechanisms for real-world problems (so that we won't use insufficient or overly expensive models). In this module, we analyze several representative problems, e.g., lexer, parser, and variable co-reference in a compiler. The real test for us is whether we can develop and describe our thought process involved in such activities and apply it to a new problem. While this is another major problem in the Theory of Computation, it is unlikely that you will easily find an answer in the literature. By all means, it is a challenge. So, you are not expected to come up with a rock solid answer, not to mention a correct answer. Do your best. As usual, you are encouraged to discuss with other students and the instructor (but your writing must be your own, reflecting your own thought, which must be unique to yourself). If we as a class come up with useful analytical tools, it would be great.

Note: Since **Tasks 1 & 2** are closely related, you should tackle them in parallel, or in a circular manner (back and forth).

**Task 1**: Develop your own "heuristics" to analyze a given problem with respect to the minimal mechanism, i.e., decidable, context-free, deterministic context-free, regular grammar/automaton/language. By "heuristics," we mean a speculative, but reasonable method of analyzing *any* given problem.

**Task 2**: This task is an application of the previous one.

(A) First, identify a new, unique practical problem (in or outside computer science) which would be a good example of demonstrating your heuristics in **Task 1**. Then, identify multiple interactive subproblems/modules of your problem. If the problem is complex, you can focus on the areas which you are most interested in. But still try to include modules of different complexity. Also, try to represent the system schematically.

(B) Next, for each subproblem/module,

1. Analyze its task and clearly present it as a problem (if possible, as a computational problem in the set notation),
2. Applying your heuristics developed in **Task 1**, identify the class of the minimal grammar or automaton that would correspond to the problem (mainly consider the extended Chomsky hierarchy: i.e., decidable, context-free, deterministic context-free, or regular), *and* justify your choice, and
3. Describe the component's specification or operation at a reasonable level.

## Part 2: Evaluation Form and Supporting Notes

Review the evaluation procedure. Then, complete your evaluation form and supporting notes. Bring them to the evaluation workshop (hard copy). Print them well in advance so that you can avoid potential problems, e.g., not being able to print just before the evaluation.

Survey: Time spent between after Unit C5 before the evaluation workshop: _______________

// End

Name: ______________________

# Module C Review Exercise, 4/1/05

This exercise is to be done in groups of two students (unless otherwise directed) assigned at this time. You need to complete the exercise within 30 minutes, but at the same time, you need to use up the time so that you can come up with a response as informative and complete as possible. While both of you in a group must agree on the general ideas, you must write your own version of the response. That is, your writing must be in your own words and not a word-by-word copy of your partner. Now, choose one of the options listed below. If your example problem in the comprehensive exercise and/or mini research problem(s) is/are similar to one of these, avoid discussing the same subject. For example, if you analyzed vision or mind as your mini research problem, you can still choose Option 1, but focus on different aspects of the system.

## Option 1: Human Being/Cat/Robot

First, analyze the functionality of a human being, cat, or robot (e.g., sensation/perception, cognition, emotion, behavior, physiology, kinesiology, etc.).

## Option 2: Operating System

First, analyze the functionality of a hypothetical operating system (not including applications such as compilers, which have been discussed frequently).

## Common procedure

Then, construct a modular system consisting of multiple interactive components. You can focus on the areas which you are most interested in; there will be no time to cover the entire system. But still try to include modules of different complexity. Also, try to represent the system schematically (i.e., with a diagram).

Next, for *each* component,

1. Analyze its task and clearly describe it as a computational problem, i.e., a language (if possible, in the set notation),
2. Using your heuristics, identify the class of minimal grammars or automata that would correspond to the problem (mainly consider the extended Chomsky hierarchy: non-TM-recognizable, TM-recognizable, decidable, context-free, deterministic context-free, regular; *exclude* context-sensitive), *and* justify your choice,
3. Describe the component's specification/operation at a reasonable level, and
4. [time permitting] Analyze the interface with other interacting modules, e.g., by comparing the input and the output of the interacting components.

**Be prepared to informally present your response to the class [approx. 5 minutes per group].**

// End

Name: ______________________

# Exercise C6/D0, 4/1/05

## Complexity (Data-Size Scalability)

During the first half of Module D, we will be discussing the third subarea of the traditional Theory of Computation, the efficiency of an algorithm with respect to the input size (data-size scalability). For example, you must remember that while a naive sorting algorithm would run in $O(n^2)$, "quick sort" runs in $O(n \log n)$. You must be familiar with this subject to some extent from other courses, most notably, Advanced Algorithms. So, try to bring in whatever you know and expand on it. As in Module C, I often notice that students' understanding of this area varies greatly. As we proceed, try to exchange what you know with other students and the instructor so that we all learn through the process.

**Task**: Write a concise essay about algorithm efficiency with respect to the input size (of course, for a decidable problem/subproblem), referring to (i) your problem from Exercise 00 and (ii) your mini research question(s) from earlier exercises or different one(s) from the list (below). In addition to discussing these problems in their entirety, you must also consider possible subproblems.

**Note**: If possible, it would be good to provide a precise asymptotic analysis, e.g., the $O$ notation (or its variants). However, if you are not comfortable doing so, try your best to describe (1) how the algorithm performance depends on the input data size and (2) the difficulty you are having with asymptotic analysis. In any case, you may want to review the materials in Advanced Algorithms.

### List of sample research questions from Exercise A6/B0

1. Can **organizational dynamics** be modeled as an algorithm?
2. Can **evolution** be modeled as an algorithm?
3. Can **ecology** be modeled as an algorithm?
4. Can **human development** be modeled as a computer?
5. Can our **minds** be modeled as a computer?
6. Can **vision** be modeled as an algorithm?
7. Can **learning** be modeled as an algorithm?
8. What would be the minimal mechanism to process **human language**?
9. Can the entire situation of an arbitrary **game** be modeled as an algorithm?
10. Would "perfect" **user modeling**, e.g., for web search, be possible?
11. Would "perfect" **computer security** be possible?
12. Can the entire process of **software engineering** be modeled computationally?
13. Can **computer networks** be modeled as a single computer?
14. Could **biology** be reduced to physics?
15. Could some computer generate real (not pseudo) **random numbers**?
16. Would it be possible to decide whether the given numbers are **random**?
17. Would **randomization** affect computability and/or complexity?
18. Would **parallelism** affect computability and/or complexity?
19. Would **artificial neural network** be more powerful than TMs?
20. Would **cellular automata** be more powerful than TMs?
21. Would the use of **analog** (or fuzzy) values affect computability?
22. Would relativistic, quantum, or some other **modern-physics**-based computation surpass TMs?

23. Can all the cases of **on-line algorithm**s be simulated by off-line computation? [On-line algorithms would obtain inputs as the time progress. Off-line computation would provide all the possibilities as input at once. Cf. function-to-relation conversion used to fold the output within the input]
24. Would **oracle computing** affect computability? [Oracle computing: A TM with the capability to ask questions to another mechanism]
25. Would **persistent TM** be able to compute more than the standard TM? [Persistent TM: Multiple sessions of TM operation with some memory between them]
26. Would **accelerating** a TM give more power?
27. Would slight **error tolerance** affect any aspect of the Theory of Computation?
28. What would be the ability of a finite automaton with a **queue**?
29. What would be the effect of "**constant**" (as in complexity analysis) in practice?
30. Can any **mathematical function** be represented computationally?
31. What exactly are **power sets** doing to the Theory of Computation?
32. Is what you can do in **logic** the same as what you can do with computation?
33. If you have a research question of **your own**, please consult the instructor first.

Survey: Time spent between classes: ______________

// End

# Unit C6 Supplement, 4/4/05

## Language and the Set Representation of a Problem

After reading some (but not all) of your Module C Comprehensive Exercises, I noticed that some of you confused the notion of "language" and "problem." Probably, I did not emphasize this point sufficiently. So, here is another supplement to clarify the connection. A language (set of strings) can always be seen as a problem (naturally, a set representation). For example, $0^n1^n = \{0^n1^n \mid n$ is a natural number$\}$ is a language and a problem at the same time. However, not all problems in set notation can be a language. For example, a problem $ACCEPT_{TM} = \{(M, w) \mid$ TM $M$ accepts string $w\}$ is not a language because a member of this problem is a "pair," not a string (even though the pair contains a string). Some problems would be represented as a set of more complex mathematical structures ("tuples" of various components). To analyze problems with respect to our extended Chomsky hierarchy, we must represent problems not just as a set, but as a language, because that is what grammars and automata are supposed to specify/process.

In some cases, it would be more natural to represent a problem using a mathematical structure as in $ACCEPT_{TM}$. In many cases, we can still modify the problem as a language. For example, by considering the string representation of the formal definition of a TM, we might represent $ACCEPT_{TM}$ as $\{M\#w \mid$ TM corresponding to the formal definition $M$ accepts string $w\}$, which can be processed by a TM. Another example would be binary addition. We can represent it as a problem (but not language), $\{(x, y, z) \mid x + y = z$, where $x, y, z$ are non-negative binary numbers (values)$\}$, or a language/problem, $\{x\#y\#z \mid val(x)+ val(y)= val(z)$, where $x, y, z$ are non-negative binary numbers (strings) whose values are evaluated with the function $val\}$, which can be processed by a TM (or possibly some other automata).

// End

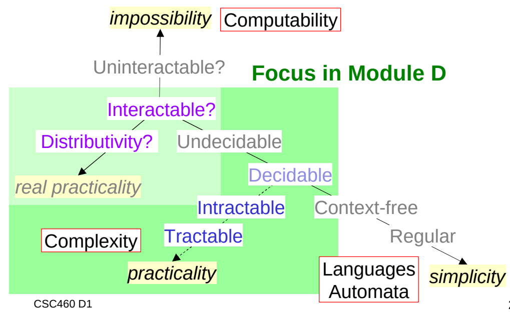
# Module C Evaluation

Review your portfolio

- My comments are sporadic and scattered on Review Ex, Comprehensive Ex, Supp. Notes, and Reflective Essay.
- You are encouraged to clarify and discuss my comments.
- You can keep the folder till the next class; then, return it to me.



# Overview: Theory of Computation






# Module D Plan

| | |
|---|---|
| D1 | Polynomial vs. Exponential<br>– Intractability<br>– Nondeterministic Polynomial-time (NP) |
| D2 | NP-Completeness (NPC) — Mini research |
| D3 | Space complexity |
| D4 | Parallel computation |
| D5/6 | Super-Turing computation — Reading |
| D7 | Evaluation workshop |
| YY | Practicum evaluation (Wed., Apr. 27) |



# Content Goal 6: Complexity

a. Reviewed how to interpret the big *O* notation [game-theoretically, using the on-line graphing tool].
b. Understood that exponential growth with respect to the input data size (time complexity) is considered impractical ("intractable"), through examples.
c. Understood the class of "nondeterministic polynomial" (NP) problems, through examples. Also understood (i) why there are so many NP problems and (ii) why NP problems are essential for computer security.
d. Understood the notion of "polynomial time reducibility" and its impact on relating problems.
e. Understood the class of "nondeterministic polynomial complete" (NPC) problems, through examples. Also understood the basics of showing that a problem is in NPC.
f. Understood the essential difference between time and space complexity, as well as the hierarchy of time/space complexity.
g. Could explain (i.e., teach) this goal to CS students outside this class.



# Content Goal 3: Interactivity

a. Understood the effects/limitations of parallel computation with respect to the three subareas of the traditional Theory of Computation.
b. Understood what kind of problems can*not* be adequately represented by TMs.
c. Understood the basics of super-Turing computation (more "powerful" than TMs) including its significance.
d. Was able to speculate where the theoretical underpinning of computer science should be heading, in order to offer robust analyses of a variety of computational problems.



# Unit D1: Overview

- Review basic ideas about algorithm analysis
- Identify the gap between practical and impractical algorithms
- Explore some questionable cases with respect to practicality
- Preview Exercise D1 "Complexity Analysis"

Thesis Penn

## How to Choose Particles in Japanese?

(in English-Japanese Translation)

Title: Osteoporosis in Active Women

(i) English: Osteoporosis is a disease of bone.

Japanese´: Osteoporosis-$\boxed{\text{wa}}$ bone-of disease-is.

(ii) English: Young females are most affected.

Japanese´: Young female-$\boxed{\text{ga}}$ most are.affected.

6/22/99 1/32 Nobo Komagata

## Parsing English Texts

- Treating Hypertension in Active Patients: Which Agents Work Best With Exercise?

  Hypertension is one of the leading risk factors for cardiovascular morbidity and mortality. Unfortunately, the percentage of those who have hypertension may be on the rise as more and more Americans remain sedentary. Physicians, therefore, need to promote exercise programs (see “Exercisers: A Healthy Minority,” below)—but without unwittingly sabotaging exercise capacity with antihypertensive medications. So in addition to knowing antihypertensive agents’ impact on exercise, clinicians need to target therapy for varied hypertensive patients who participate in different sports and types of exercise.



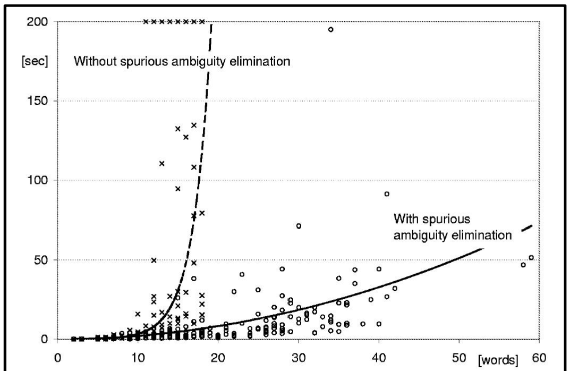



Figure 2: English Parser with and without Spurious Ambiguity Elimination

## Practicality

- Where can we draw a line between *practical* and *impractical* algorithms, esp. facing *realistically large data*?
- Focus on the effect of input data size



## Digital Circuits (Sample Problem #13)

- To design digital logic circuits, we need to analyze the relation between inputs and the output. For example, the following formula represents a circuit with four inputs and one output, each of which could take either 0 or 1 (‘+’ for OR, ‘·’ for AND, and ‘'’ for NOT).

  output = $(i_1' + i_2 + i_3 + i_4) \cdot (i_1' + i_2 + i_3 + i_4') \cdot (i_1' + i_2' + i_3 + i_4')$
- Brute-force algorithm?



## Cross-Country Interview (Sample Problem #16)

- Suppose that you are invited for interviews (graduate schools or industry jobs) in a number of cities in the US. Unfortunately, they do not pay for your travel. So, you will need to minimize the expense.
- Simple-minded algorithm?

## Faster Algorithms

- Count from 0 to the user-specified $n$

```
n <- input
for (int i = 0; i < n; ++i) {
    output <- i;
}
```

- Display elements of a $n \times m$ matrix

```
n <- input; m <- input; // set elements in matrix
for (int i = 0; i < n; ++i) {
    for (int j = 0; j < m; ++j) {
        output <- matrix[i, j];
    }
}
```



## Time Complexity

(asymptotic analysis)

- The speed/time performance of an algorithm as a function of the input data size, *ignoring the constant factor*
- Can be analyzed with the big-$O$ notation (and its variants)



## Big-$O$ Notation

- Informal idea: Behaves roughly like such an such function popular use of '=' instead of '∈'
- Definition #1: $f(n) \in O(g(n))$ if there are constants $c > 0$ and $n_0 \geq 1$ such that $f(n) \leq c\, g(n)$ for every integer $n \geq n_0$.
- Definition #2:

$$f(n) \in O(g(n)) \quad \text{if } \lim_{n\to\infty} \frac{f(n)}{g(n)} = c \quad \text{for some } c \geq 0$$



## Group Exercise 1

A. Explain why $n^2 \in O(2^n)$

B. Explain why $2^n \notin O(n^2)$

- Definition #1: $f(n) \in O(g(n))$ if there are constants $c > 0$ and $n_0 \geq 1$ such that $f(n) \leq c$ $g(n)$ for every integer $n \geq n_0$.
- Definition #2:

$$f(n) \in O(g(n)) \quad \text{if } \lim_{n\to\infty} \frac{f(n)}{g(n)} = c \quad \text{for some } c \geq 0$$



## Common Time Complexity Classes

- $O(\log n)$ = Logarithmic
- $O(n)$ = Linear
- $O(n \log n)$
- $O(n^2)$ = Quadratic
- $O(n^3)$ = Cubic
- Note: $O(n^i) \subseteq O(n^j)$ where $i \leq j$

Examples?

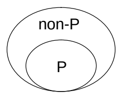


## Polynomial vs. Exponential

- Polynomial-Time (Bound) Problems
  P = $O(n^k)$ for any constant $k \geq 1$
  - Includes faster algorithms as well: $O(\log n) \subseteq O(n \log n) \subseteq$ P
- Exponential-Time (Bound) Problems
  Exp = $O(c^n)$ for any constant $c > 1$
  - Includes faster algorithms as well: P $\subseteq$ Exp
- (strictly) Exponential problems: Exp – P

## Intractability

- Tractable problems = P
- Intractable problems = Complement of P
  - Including exponential problems (brute-force algorithms are not acceptable except for toy examples)




Is the gap clear cut?



## Sample Problems, Revisited

- Digital Circuits = SATISFIABILITY (SAT)
  - Example: (*p* **or** *q*) **and** (*p* **or** (**not** *q*)) **and** ((**not** *p*) **or** *r*)
  - Search space: Exponential
  - Practical algorithm?
- Cross-country interviews = TRAVELING SALESPERSON (TSP)
  - Search space: Exponential
  - Practical algorithm?



## SAT & TSP: Properties

- Search space: Exponential
- Exhaustive search $\Rightarrow$ Exponential
- Checking an answer $\Rightarrow$ Linear
- No polynomial algorithm has been found. $\Rightarrow$ Most likely intractable
  - But nobody has proved the nonexistence. $\Rightarrow$ Could be tractable
- Practical approach $\Rightarrow$ Approximate



## Approximate Solutions

- Classical local search (SAT, TSP)
- Greedy algorithm (SAT, TSP)
- Dynamic programming (TSP)
- Simulated annealing (SAT)
- Genetic algorithm (SAT, TSP)
- Neural network (TSP)

*How to Solve It: Modern Heuristics* by Michalewicz and Fogel



## Nondeterminism

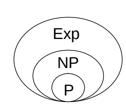
- Possibility of exponential branching
- Does not change the power of TMs
- Could change the time complexity of processing certain NP problems

cf. simulation of nondeterministic TM



## Class NP

- Class of Nondeterministic polynomial-time problems
  - E.g., SAT, TSP, and many others
- Definition #1: A solution can be *verified* in polynomial time.
- Definition #2: A nondeterministic TM can decide in polynomial time.




Connection to computer security?

## Group Exercise 2

A. Show that a scheduling problem (find a combination that satisfy some constraint) is in NP, using the both definitions.
B. Is a NP problem tractable? Explain?

•Definition #1: A solution can be *verified* in polynomial time.

•Definition #2: A nondeterministic TM can decide in polynomial time.



## Sample NP Problems

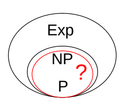
- Cryptography (Sample Problem #11)
- Monkey Puzzle (#12)
- Professor Assignment (#14) ~ scheduling problem
- Knapsack Problem (#15)
- CPU Register Allocation (#17)
- Map Coloring (#18)

Why so many?



## Unit Summary

- Use of the big-$O$ notation for analyzing time complexity
- Hierarchy of time complexity classes: P (tractable), NP (open), Exp (intractable)
  - No polynomial algorithms have been found for any of the NP problems.
  - P = NP?






## Summary Question

- Do you understand the definition of NP?
  - Don't leave until your answer becomes yes.
- Questions/Comments/Suggestions

# Unit D1 Supplement, 4/8/05

## Sample Response to Exercise D1 Part 1: Big-*O* Notation

This exercise is intended to serve not only as a review of algorithm analysis but also as a review of logic. The use of game-theoretic interpretation often reveals whether one understands the semantic aspect of logic (which is more important than the syntactic aspect, in my opinion). If one can associate quantifiers with the operations of the graphing tool, s/he clearly demonstrates how to deal with quantifiers. I realized that most of you need to see how to do it. So, here is how I would do.

**Definition**: '*O*' (asymptotic upper bound) is defined as follows: $f(n) \in O(g(n))$ if there are constants $c > 0$ and $n_0 \geq 1$ such that $f(n) \leq c\, g(n)$ for every integer $n \geq n_0$.

The definition can be translated into the following first-order logic (FOL) statement (use of '$\wedge$' for 'and'):

$$\exists c\ \exists n_0\ \forall n\ [\ (c > 0) \wedge (n_0 \geq 1) \wedge (n \geq n_0) \wedge (f(n) \leq c\, g(n))\ ]$$

which can also be written as: $\exists c > 0\ \exists n_0 \geq 1\ \forall n \geq n_0\ [\ f(n) \leq c\, g(n)\ ]$. As for the ordering of the quantifier, $\exists n_0$ and $\forall n$ must be placed in that order because the choice of $n$ depends on that of $n_0$. The placement of $\exists c$ is flexible because only the main relation, $f(n) \leq c\, g(n)$, depends on it.

Advanced notes (can be skipped): Although it would also be possible to quantify the functions as follows:

$$\forall f\ \forall g\ \exists c\ \exists n_0\ \forall n\ [\ (c > 0) \wedge (n_0 \geq 1) \wedge (n \geq n_0) \wedge (f(n) \leq c\, g(n))\ ]$$

such a statement is beyond the ability of FOL because functions are "higher-order" and not individuals.

The general procedure would be as follows. First, assume that the two functions are given. Then, define a strategy for each quantifier.

1. Verifier: Choose $c > 0$ by adjusting the slide bar so that the statement would hold.
2. Verifier: Choose $n_0 \geq 1$ by finding the value by adjusting the scale(s) so that the statement would hold.
3. Falsifier: Choose $n \geq n_0$ by finding the value by adjusting the scale(s) so that the statement would not hold.

If the falsifier can find such a value, the falsifier wins, i.e., the statement does not hold. Otherwise, the verifier wins, i.e., the statement holds.

Example 1: $n^4 \in O(2^n)$

- Set $f(n) = n^4$, $g(n) = 2^n$ Draw the graph with *x*-axis: 10^1, *y*-axis: 10^3 (default); $f(n) = n^4$ looks like growing faster.

- The verifier keeps the default $c = 1$, but searches for a $n_0$ by adjusting $x$-axis to 10^2. Since the values go beyond the current range, s/he increases the $y$-axis to 10^6. Since $g(n) = 2^n$ appears to exceed $f(n) = n^4$ after $n = 16$ (or so), the verifier sets the $n_0$ to 20 (to be safe).
- The falsifier tries to find an $n \geq n_0$ at which the situation reverses by extending the $x$-axis (and $y$-axis to increase the range). However, s/he fails to find such an $n$. The falsifier lost. Thus, $n^4 \in O(2^n)$.

Example 1: $2^n \notin O(n^2)$

- Set $f(n) = 2^n$, $g(n) = n^2$ Draw the graph with $x$-axis: 10^1, $y$-axis: 10^3 (default); $f(n) = 2^n$ looks like growing faster.
- The verifier tries to raise $c = 10$. Then, $g(n) = n^2$ appears be no less than $f(n) = 2^n$, within the window. Just to be make $c\,g(n)$ larger, s/he raises $c$ to 10,000. S/he stays with $n_0 = 1$.
- But then, the falsifier can adjust the scale to observe that $f(n) = 2^n$ exceed $g(n) = n^2$ at some point. Although this step depends on the verifier's choice of $c$ and $n_0$, the falsifier seems to be able to find a point where $f(n) = 2^n$ exceed $g(n) = n^2$, at least within the operations of the graphing tools. The falsifier won. Thus, $2^n \notin O(n^2)$.

// End

Name: ______________________

# Exercise D1, 4/5/05

## Part 1: Big-*O* Notation

The big-*O* notation is useful for analyzing the effect of the input data size on the performance of an algorithm (not including the constant factor corresponding to, e.g., fixed amount of speed up). Although this concept must have been discussed in other courses, e.g., Advanced Algorithms, it would still be good to review the idea. To do so, we will re-use some of the tools introduced earlier in this course, i.e., game-theoretic interpretation of FOL and the on-line graphing tool. First, here is a definition of the big-*O* notation.

**Definition**: '*O*' (asymptotic upper bound) is defined as follows: $f(n) \in O(g(n))$ if there are constants $c > 0$ and $n_0 \geq 1$ such that $f(n) \leq c\, g(n)$ for every integer $n \geq n_0$.

**Task**: Give a game-theoretic interpretation of the above definition *referring to the operation of the on-line graphing tool* (http://www.tcnj.edu/~komagata/Graphing). Note that you can adjust the constant $c$ using a slide bar (and a text field) and designate some $n_0$ by simply identifying a specific value on the $x$ axis (note that you can change the graph scale).

Hint: If you are not sure how to use the tool, read the tutorial part of the documentation linked from the graphing tool page.

## Part 2: Sample Problems

As discussed in class, there are a large number of NP problems. NP problems are important because many practical *combinatory* problems exhibit the properties of this class.

**Task**: Choose at least two problems from the Sample Problems (http://www.tcnj.edu/~komagata/csc460/05s/SampleProblems.pdf) listed below, and informally explain why the problems are in the class NP.

- #11 Cryptography
- #12 Monkey Puzzle
- #15 Knapsack Problem
- #17 CPU Register Allocation
- #18 Map Coloring

Survey: Time spent between classes: ______________

// End

## Questions

- How good is computer security?
- What would computer security professionals and hackers pursue?



## Unit D2: Overview

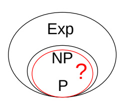
- Notice the connection among NP problems
- Understand polynomial time reducibility
- Identify and analyze a special class of NP problems
- Preview Exercise D2 “NP/NPC Practice; Mini Research Ideas”



Review

## NP Problems

- Class of Nondeterministic Polynomial-time problems
  - E.g., SAT, TSP, and many others
- Definition #1: A solution can be verified in polynomial time.
- Definition #2: A nondeterministic TM can decide in polynomial time.






Review

## Sample NP Problems

- Digital Circuits (#13) = SATISFIABILITY (SAT)
- Cross-country interviews (#16) = TRAVELING SALESPERSON (TSP)
- Cryptography (#11)
- Monkey Puzzle (#12)
- Professor Assignment (#14)
- Knapsack Problem (#15)
- CPU Register Allocation (#17)
- Map Coloring (#18)



## Standard NP Problems

- SATISFIABILITY (SAT)
- 3SAT
- TRAVELING SALEPERSON (TSP)
- HAMILTONIAN CIRCUIT (HC)
- VERTEX COVER (VC)
- CLIQUE
- 3D MATCHING (3DM)
- PARTITION



## SATISFIABILITY (SAT)

- Informally, finding a satisfying truth-value assignment for a conjunction of (disjunctive) clauses, e.g.,
  $(p \text{ or } q) \text{ and } (p \text{ or } (\text{not } q)) \text{ and } \overbrace{((\text{not } p) \text{ or } r \text{ or } s)}^{\text{clause}}$.
  - Note: A clause is a disjunction of propositional (true/false) variables or its negation.
- Formally, $\{(C, V, a) \mid$ set of (disjunctive) clauses $C$ with a finite set of variables $V$ such that assignment $a$: $V \rightarrow \{\mathbf{T}, \mathbf{F}\}$ satisfies *all* the clauses in $C\}$

## 3SAT

- A special case of SAT with the pattern

  **(_ or _ or _) and (_ or _ or _) and ... and (_ or _ or _)**

  where each '_' holds a variable (e.g., $\boldsymbol{p}$) or the negation of a variable (e.g., **not** $\boldsymbol{q}$)
  – I.e., each (...) is limited to 3 things
- Connection to SAT?



## TRAVELING SALESPERSON (TSP)

- Informally, finding a tour of all the cities (without repeating) within the set budget.
- Formally, $\{(\mathbf{G}, c, b) \mid$ graph $\mathbf{G} = (V, E)$ with the cost function $c$ such that a tour is possible within the total cost of $b\}$
  - tour: $(v_1, ..., v_{k=|V|})$ such that
    $v_i \in V$ for every $1 \le i \le k$,
    $v_i \ne v_j$ for every pair $i \ne j$,
    $(v_i, v_{i+1}) \in E$ for every $1 \le i < k$,
    $(v_k, v_1) \in E$
    $c: V \times V \to \mathbf{Z}^+$ (positive integer)

completeness of $\mathbf{G}$?



## HAMILTONIAN CIRCUIT (HC)

- Informally, finding a closed circuit to travel all the points without repeating.
- Formally, $\{\mathbf{G} \mid$ graph $\mathbf{G} = (V, E)$ admits a hamiltonian circuit$\}$
- hamiltonian circuit: $(v_1, ..., v_{k=|V|})$ such that
  $v_i \in V$ for every $1 \le i \le k$,
  $v_i \ne v_j$ for every pair $i \ne j$,
  $(v_i, v_{i+1}) \in E$ for every $1 \le i < k$,
  $(v_k, v_1) \in E$
- Connection to TSP?

Hamiltonian *Path* ~ between two points
Euler circuit ~ traversing all the edges



## VERTEX COVER (VC)

- Informally, finding a graph segment of size $k$ or less that contain at least one point of every edge.
- Formally, $\{(\mathbf{G}, k) \mid$ graph $\mathbf{G} = (V, E)$ admits a vertex cover where $0 < k \le |V|\}$
- vertex cover: $W \subseteq V$ such that
  $|W| \le k$,
  For every $(u, v) \in E$, $u \in W$ or $v \in W$



## CLIQUE

- Informally, finding a subgraph of size $k$ or less such that every two nodes are connected by an edge in the graph.
- Formally, $\{(\mathbf{G}, k) \mid$ graph $\mathbf{G} = (V, E)$ contains a $k$-clique where $0 < k \le |V|\}$
- $k$-clique: $W \subseteq V$ such that
  $|W| = k$,
  For every $u, v \in W$, $(u, v) \in E$
- Connection to VC?



Intermission

## Marriage Problem

- Informally, finding a complete list of male-female pairs (without polygamy)
- Formally, $\{(R, k) \mid R \subseteq M \times F$ contains a matching where $M, F$ are disjoint and $|M| = |F| = k\}$
- matching: $P \subseteq R$ such that
  $|P| = k$,
  $\forall (x, y), (u, v) \in P \ [(x \ne u) \wedge (y \ne v)]$
  (No two elements of $P$ agree in any coordinate)

NP?

# 3D MATCHING (3DM)

- Informally, a variant of the marriage problem involving 3 different sexes.
- Formally, $\{(R, k) \mid R \subseteq W \times X \times Y$ contains a matching where $W, X, Y$ are disjoint and $|W| = |X| = |Y| = k\}$
- matching: $M \subseteq R$ such that

  $|M| = k$,

  $\forall (x, y, z), (u, v, w) \in M \ \ [(x \neq u) \wedge (y \neq v) \wedge (z \neq w)]$

  (No two elements of $M$ agree in any coordinate)



# PARTITION

- Informally, finding a subset that divides a set evenly with respect to the members' "prices."
- Formally, $\{(A, s) \mid$ finite set $A$ with a balanced partition$\}$
- balanced partition: $B \subseteq A$ such that

  $p: A \rightarrow \mathbf{Z}^+$ ("price" function)

  $$\sum_{a \in B} p(a) = \sum_{a \in (A-B)} p(a)$$



connection among problems?



# Connection among Problems

- Special cases
  - 3SAT (clauses of size 3) is a special case of SAT
  - HC is a special case of TSP (with weight)
- Similarity
  - VC and CLIQUE
- Can some problems be reduced to others?

Requirements for reduction?



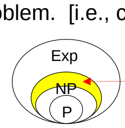
# Polynomial Time Reducibility

- $A$ is *polynomial time* reducible (P-reducible) to $B$.
  - $A$ is reducible to $B$.
  - This can be done in polynomial time.
- Transfer of properties
  - $B$ has a positive property $\Rightarrow$ $A$ has it too.
    - Example: If $B \in$ P, $A \in$ P.
  - $A$ has a negative property $\Rightarrow$ $B$ has it too.
    - Example: If $A \notin$ P, $B \notin$ P.



# NP-Complete (NPC) Problems

- Definition
  1. The problem is in NP.
  2. *Any* NP problem can be P-reduced to *that* problem. [i.e., can solve any NP problem]




NPC

Most difficult NP problems

- Generalization of X-complete
  - P-complete, Turing-complete, AI-complete



# Cook-Levin Theorem

- SAT is NP-complete.
  1. SAT is in NP.
  2. Any NP problem can be P-reduced to SAT.
- Challenge
  - Generic NP problem representation
- Terminology

$$\underbrace{(\mathbf{p} \text{ or } \mathbf{q})}_{\text{clause}} \text{ and } \underbrace{(\mathbf{p} \text{ or } (\text{not } \mathbf{q}))}_{\text{clause}} \text{ and } \underbrace{((\text{not } \mathbf{p}) \text{ or } \mathbf{r})}_{\text{clause}}$$

Cook-Levin Theorem

## Proof Outline

- Lemma: Any NP problem can be P-reduced to SAT.
- Components
  - An arbitrary polynomial time Nondeterministic TM (NTM) program (or verification sequence of an answer) can be reduced to SAT.
    - Acceptable strings → True SAT statements
    - Acceptable strings ← True SAT statements
  - The reduction is in P (polynomial reduction).
    - Complexity analysis

Garey & Johnson 1979 (standard NPC reference)



Cook-Levin Theorem

## Polynomial Time Conditions

- Cannot spend exponential amount of time (cf. the input)
- Cannot use an exponential amount of tape space
  - The limit to polynomial tape space does not guarantee P. But exponential space cannot be handled in P.
- Cannot use an exponential number of symbols



Cook-Levin Theorem

## Reduction Idea

Introduce *propositional variables* as follows:

- Simulating states
  – TM is in state $q_k$ at time $i \Rightarrow Q_{(i,k)}$
- Simulating head movements
  – Head is at tape position $j$ at time $i \Rightarrow H_{(i,j)}$
- Simulating tape symbols
  – Tape symbol of position $j$ is $s_k$ at time $i$ $\Rightarrow S_{(i,j,k)}$

Polynomially bounded?



Cook-Levin Theorem

## Imposing Logical Conditions

- At time **0**, TM is in its initial configuration.
- At time $i$, TM
  – Is in exactly one state,
  – Is at exactly one tape position, and
  – Reads exactly one tape symbol, and
  – Moves to the next configuration defined by d.
- Within a polynomially-bounded time, TM enters the final configuration.

view as verification with a NTM

- States $\Rightarrow Q_{(i,k)}$
- Head movements $\Rightarrow H_{(i,j)}$
- Tape symbols $\Rightarrow S_{(i,j,k)}$



## 3SAT Statements

- States $\Rightarrow Q_{(i,k)}$
- Head movements $\Rightarrow H_{(i,j)}$
- Tape symbols $\Rightarrow S_{(i,j,k)}$
- '∧' = 'and', '∨' = 'or', '¬' = 'not'

- At time 0, TM is in its initial configuration. $\Rightarrow Q_{(0,0)} \wedge H_{(0,0)} \wedge S_{(0,0,k_0)} \wedge S_{(0,1,k_1)} \wedge \ldots \wedge S_{(0,n-1,k_{n-1})} \wedge S_{(0,n,0)} \wedge \ldots \wedge S_{(0,\underline{\text{poly-bound}},0)}$
- At time $i$, TM
  - Is in exactly one state ⇒ at least two & at most one ⇒ $(Q_{(i,0)} \vee \ldots \vee Q_{(i,\underline{\text{poly-bound}})}) \wedge \neg(Q_{(i,j)} \wedge Q_{(i,j')}) \wedge \ldots$ [for all $j \neq j'$]
  - Moves to a next configuration defined by δ.
    - Case 1 (at $j$ at time $i$):
      $((H_{(i,j)} \wedge Q_{(i,k)} \wedge S_{(i,j,l)}) \rightarrow H_{(i+1,j\ +/-\text{movement})}) \wedge (\ldots \vee Q_{(i+1,k')}) \wedge (\ldots \vee S_{(i+1,j,l')})$
      Note $((H_{(i,j)} \wedge Q_{(i,k)} \wedge S_{(i,j,l)}) \rightarrow H_{(i+1,+/-\text{movement})}) \leftrightarrow (\neg H_{(i,j)} \vee \neg Q_{(i,k)} \vee \neg S_{(i,j,l)} \vee H_{(i+1,j+\text{some value})})$
    - Case 2 (not at $j$ at time $i$): i.e., no change
      $((\neg H_{(i,j)} \wedge S_{(i,j,k)}) \rightarrow S_{(i+1,j,k)})$
      Note $((S_{(i,j,k)} \wedge \neg H_{(i,j)}) \rightarrow S_{(i+1,j,k)}) \leftrightarrow (H_{(i,j)} \vee \neg S_{(i,j,k)} \vee S_{(i+1,j,k)})$

all clauses?



## Cook-Levin Theorem Summary

SAT is NP-complete.

1. SAT is in NP. ⇐ Truth value evaluation
2. Any NP problem can be P-reduced to SAT.
   - An arbitrary NTM program in P is polynomial time reducible to SAT.
     - Simulate states, head movements, and tape symbols as a logical condition
     - Construct the statement in polynomial time

## 3SAT is NPC

- 3SAT is in NP.
- SAT is polynomially reducible to 3SAT.
  - If 3SAT can solve SAT, 3SAT is NPC.
- Reduction idea
  - Transform each clause into an equivalent collection of clauses with 3 variables
  - Computability: Existence of an algorithm
  - Tractability: Polynomial bound

How about 2SAT?



## VC is NPC

- 3SAT is polynomially reducible to VC.
- Construct a graph with some $k$ that would translate 3SAT to VC
- Example

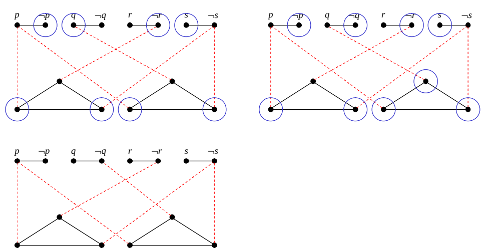
  **(*p* or (not *r*) or (not *s*)) and (*p* or *q* or (not *s*))**

  **(*p* Ú Ø*r* Ú Ø*s*) Ù (*p* Ú *q* Ú Ø*s*)**

  OK to use this special VC?
- Polynomial reduction to CLIQUE possible



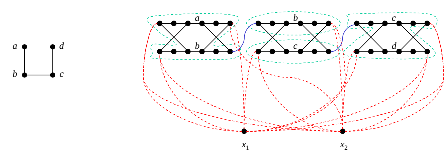
## 3SAT to VC





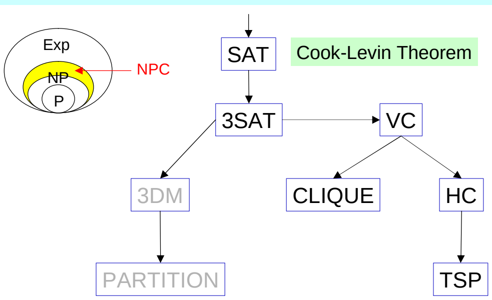
## HC is NPC

- VC is polynomially reducible to HC.
- ... a bit complicated...





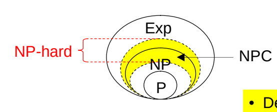
## NPC Summary






## NP-Hard Problems

- Definition
  - Any NP problem can be reduced to the problem. [at least as difficult as NPC]
  - Problem is *not necessarily* in NP. [OK to be in NP ⇒ NP-hard includes NPC]




*X*-Hard for any class *X*

- Define NPC with NP-hard
- Type of acceptable reductions?

## Unit Summary

- A whole bunch of practical problems are in NPC.
- All NPC problems are equivalent with respect to their time complexity via polynomial reduction.
- A polynomial solution to any of these NPC problem would conclude P = NP.



## Summary Question

- Do you understand the significance of NPC problems? Explain.
- Questions/Comments/Suggestions

Name: ______________________

# Exercise D2, 4/8/05

## Part 1: MAX-CLIQUE

In the future, you will encounter new problems. In many cases, you will not know what the status of the problem is (unlike many school examples). So, it's good to be prepared for such a situation. Here is a new problem that is related to CLIQUE.

Let MAX-CLIQUE = {($G$, $k$) | the *largest* clique of a graph $G$ = ($V$, $E$) has $k$ vertices}. <u>Whether this problem is in NP is unknown.</u>

A. Explain what the underlined statement above mean.
B. Speculate whether this problem is in NP. Try to justify.

## Part 2: 3COLOR

Knowing that a problem is in NPC is a mixed message. However, with that information, we can be prepared to deal with the problem. Until there is a conclusion, we need to live with impractical exhaustive search or some practical approximate algorithms.

Let us define a problem 3COLOR as {$G$ | the nodes of a graph $G$ = ($V$, $E$) can be colored with three colors such that no two nodes joined by an edge have the same color}.

A. Show that 3COLOR is in NP.
B. 3COLOR is actually in NPC. Explain how you would show that it's in NPC. Note that you do ***not*** need to prove that 3COLOR is in NPC. However, you must identify all the necessary information that you would need to prove it.

## Part 3: Mini Research Ideas

As the first step to tackle Module D Comprehensive Exercise, you will narrow down on the topic and specific research question.

**Task**: Read the Module D Comprehensive Exercise carefully. Then, identify a mini research question which is related to the content of this course. If you feel that you have adequately solved a problem in earlier exercises, come up with a new problem to which you do not know the answer.

Survey: Time spent between classes: ______________

// End

## Code Talker & Chomsky

- Who are "code talkers"?
- What would Chomsky say about them?
- Anything to do with time complexity?



## Ex C2

Part 1: MAX-CLIQUE

- {(*G*, *k*) | the largest clique of *G* has *k* vertices}
- Explain: "Unknown if it is in NP."
- Speculate

Part 2: 3COLOR

- {*G* | the nodes of $G = (V, E)$ can be distinctively colored with three colors}
- 3COLOR $\in$ NP
- How to show 3COLOR $\in$ NPC?



## Mini Research Ideas



## Unit D3: Overview

- Compare time and space complexity
- Analyze the effect of nondeterminism with respect to space complexity
- Preview Exercise D3 "Mini Research Paper Outline"
  - Different types of exercises from this time on



## Time vs. Space

- They are not so different [paraphrase of Einstein].
- Time can be interpreted in another domain [applying Fourier transformation].
- One cannot be certain about both position (space) and momentum (time dimension involved ... velocity) [Heisenberg's Uncertainty Principle].
- Computers run faster with more memory [computer industry].



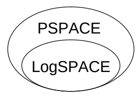
## Time vs. Space Complexity

- Time complexity
  - Maximum number of TM *steps* as a function of the input size (asymptotically)
- Space complexity
  - Maximum number of TM *tape space* as a function of the input size (asymptotically)

# Space Complexity Hierarchy

- Constant space
  - Any use?
- LogSPACE (also called L)
  - Handling inputs ... isn't it at least linear?
- LinearSPACE
  - SAT (an exhaustive algorithm)
- Polynomial space (PSPACE)
  - Examples?

Focus in this unit






# $0^n1^n$

- Review: Time complexity?
- Naive algorithm: LinSPACE
  - Usual use of a stack
- A LogSPACE algorithm possible?



# QBF Problem

- Quantified Boolean Formula

  **For all** $x$**, some** $y$ **((**$x$ **or (not** $y$**)) and ((not** $x$**) or** $y$**))**

  " $x$ \$$y$ (($x$ Ú Ø$y$) Ù (Ø$x$ Ú $y$))    Satisfiable?
  - Where each variable is either true or false
  - Cf. game-theoretic interpretation of FOL
- QBF Problem: { φ | φ is a true Quantified Boolean formula}
- Review: Time complexity?
- Space complexity?



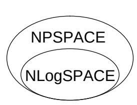
# MAZE

- Finding a path between two points
  - Also known as graph-reachability problem
- Review: Time complexity?
- Space complexity?



# Nondeterministic Space

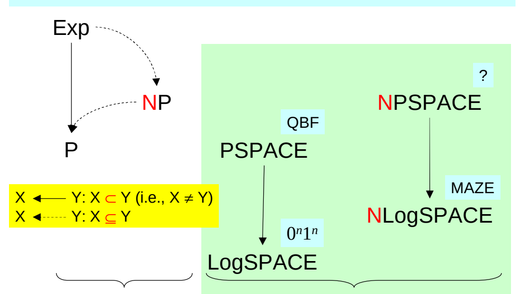
- Nondeterministic space complexity
  - Maximum number of NTM *tape space* as a function of the input size
  - Space measure for the successful run
- NLogSPACE (also called NL)
  - MAZE
- NPSPACE
  - Example?






# Complexity Hierarchy

Preview



## Today's Questions

1. *X* vs. *X*SPACE
   - E.g., P vs. PSPACE, NP vs. NPSPACE
2. *X*SPACE vs. N*X*SPACE
   - E.g., PSPACE vs. NPSPACE, LogSPACE vs. NLogSPACE
3. *X* vs. N*Log(X)*SPACE
   - E.g., Exp vs. NPSPACE, P vs. NLogSPACE



## Group Exercise: Question 1

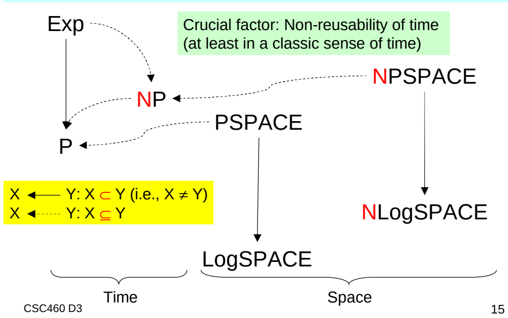
- *X* vs. *X*SPACE
  - E.g., P vs. PSPACE, NP vs. NPSPACE
- Which would be correct? Why?

A. $X \subset X$SPACE
B. $X \subseteq X$SPACE
C. $X = X$SPACE
D. $X \supseteq X$SPACE
E. $X \supset X$SPACE



Rev. 1

## Complexity Hierarchy






## Group Exercise: Question 2

- *X*SPACE vs. N*X*SPACE
  - E.g., PSPACE vs. NPSPACE, LogSPACE vs. NLogSPACE
- Which would be correct? Why?

A. *X*SPACE $\subseteq$ N*X*SPACE
B. *X*SPACE = N*X*SPACE
C. *X*SPACE $\supseteq$ N*X*SPACE
D. None of the above



## Effects of Nondeterminism

Observation so far

- $L$(TM) = $L$(NTM)
- $L$(DFA) = $L$(NFA)
- $L$(DPDA) $\subseteq$ $L$(PDA) PDAs are nondeterministic
- P $\subseteq$ NP
  - Probably P $\subset$ NP (i.e., P $\neq$ NP)



## Group Exercise: Question 3

- *X* vs. N*Log(X)*SPACE
  - E.g., Exp vs. (N)PSPACE, P vs. NLogSPACE
- Which would be correct? Why?

A. $X \subseteq$ N*Log(X)*SPACE
B. $X =$ N*Log(X)*SPACE
C. $X \supseteq$ N*Log(X)*SPACE
D. None of the above

## PSPACE-Complete

- Definition: PSPACE-complete
  1. The problem is in PSPACE.
  2. Every PSPACE problem is polynomial time reducible to the problem.
- Theorem: QBF is PSPACE-complete.
- Proof idea
  1. Existence of a PSPACE algorithm
  2. Simulate polynomial-time TM (with a technique similar to the proof of Savitch's Theorem)



## Unit Summary

- Time and space complexities have some connection.
- The effect of nondeterminism differ in each case.
- Applications of complexity theory
  - NPC and beyond ⇒ Use of approximation algorithms
  - Effect of limited resources (of different types)



## Summary Question

- What is your sense of time-space tradeoff, in the context of your choice (CS or else)? Discuss freely, referring to some example.
- Questions/Comments/Suggestions

# Unit D3 Supplement, 4/13/05

## QBF Problem

The quantified variables in a QBF statement, i.e., $x$, $y$, $z$, ..., can refer to any propositional variables. But since a propositional variable can take either true or false, we can analyze the truth-value assignment of a QBF statement with respect to possible values of true or false. Let us consider the example introduced in class:

For all $x$, some $y$ (($x$ or (not $y$)) and ((not $x$) or $y$))
[formally] $\forall x\ \exists y\ ((x \vee \neg y) \wedge (\neg x \vee y))$

To check $\forall x$, we will need to consider both $x$ = true and $x$ = false. Let us first set $x$ = true. Then, if we choose $y$ = true, the entire statement can be made true. Next, let $x$ = false. Then, we can choose $y$ = false to make the entire statement true. Thus, this statement holds. The strategy of choosing values is analogous to the game-theoretic interpretation of FOL (although in FOL, we choose an individual or value in a more general sense, not true/false).

When all the quantifiers are existential and the rest of the statement is organized as a conjunction of (disjunctive) clauses, the problem degenerates to SAT. So, if we can solve QBF, we can solve SAT. Thus, SAT reduces to QBF.

**QBF is NP-hard** (i.e., it can solve any NP problem). SAT can be reduced to QBF. Since SAT is in NPC, QBF must at least as difficult as SAT. So, any problem that SAT can solve must be solved by QBF. Note that the (positive) property that SAT is in NP does not transfer to QBF.

However, in the worst case, all the quantifiers can be universal. Then, we need to check all the combinations of truth values. So, even to verify an answer, it will take exponential time, as shown in an example with 3 variables below.

| Combo | $x_1$ | $x_2$ | $x_3$ |
|---|---|---|---|
| 1 | T | T | T |
| 2 | T | T | F |
| 3 | T | F | T |
| 4 | T | F | F |
| 5 | F | T | T |
| 6 | F | T | F |
| 7 | F | F | T |
| 8 | F | F | F |

**QBF is not in NP**. Due to the quantifiers (the universal quantifier in particular), it is not possible to verify the truth value of a given statement in polynomial time. That is, even to verify a correct answer, we will still need to check all the combinations of truth-value assignment. Thus, it is worse than NP.

**QBF is in PSPACE**. Given any QBF statement, we can analyze its truth value in a way analogous to the analysis of the example statement above. Suppose that there are three universally quantified variables, $x_1$, $x_2$, and $x_3$. First, set $x_1$ = true in the working tape position 1. Then, set $x_2$ = true in the working tape position 2, and so on, as shown schematically below.

| | | |
|---|---|---|
| $x_1$ | $x_2$ | $x_3$ |

If this assignment result in false, the entire statement cannot be true (because all the combinations must result in true). Using a recursion, we can cover all the combinations. So, we need $O(n)$ working tape space, which is Polynomial.

## Time-Space Tradeoff

One natural idea about time-space tradeoff may be that with more space, one can reduce the time. Scott (in his summary question) represented this (very roughly) as: Cost = Time $\times$ Space. Consider the following example. You open a computer repair shop in your own dorm room. If ten customers bring in ten computers, your work efficiency may well suffer from the lack of space. On the other hand, if you had a typical classroom, you could do various things much more efficiently. The time-space tradeoff for computers is often discussed along this line.

Departing from this particular task, let us compare your dorm room and a typical classroom with respect to a broader range of tasks. If you need to vacuum, you will need more time for the classroom. If you need to keep the space warm, it will cost more to warm up the entire classroom. The discussion of the relation P $\subseteq$ PSPACE appears to be more related to this view. That is, with more space, you could potentially do more things, which would potentially take more time.

// End

Name: ______________________

# Exercise D3, 4/12/05

## Mini Research Outline

Starting from the ideas you brought in Exercise D2 Part 3, you will develop the backbone of your mini research paper. The most important part at this stage is to identify your research question. Then, analyze its cost/significance and speculate how you would respond to the question. At that point, you may well revisit and revise your research question. Note that it is a common practice even for an experienced researcher to iterate this process.

**Task**: Write up the outline of your mini research paper. Try to respond to each of the bullets in Module D Comprehensive Exercise Part 1 Task 1.

Survey: Time spent between classes: ______________

// End

## Mini Research

- Questions?
- Status?

Practicum evaluation
- Wed., Apr. 27
- Most of you are assigned earlier presentations 12:00pm ~
- Detailed instructions will be distributed later (but already available on-line)



Unit D3 (cont'd)

## Today's Questions

1. *X* vs. *X*SPACE
   - E.g., P vs. PSPACE, NP vs. NPSPACE
2. *X*SPACE vs. N*X*SPACE
   - E.g., PSPACE vs. NPSPACE, LogSPACE vs. NLogSPACE
3. *X* vs. N*Log(X)*SPACE
   - E.g., Exp vs. NPSPACE, P vs. NLogSPACE

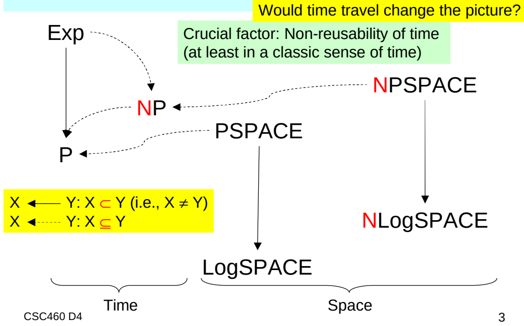


Rev. 1

## Complexity Hierarchy

Would time travel change the picture?

Crucial factor: Non-reusability of time (at least in a classic sense of time)






## Group Exercise: Question 2

- *X*SPACE vs. N*X*SPACE
  - E.g., PSPACE vs. NPSPACE, LogSPACE vs. NLogSPACE
- Which would be correct? Why?

A. *X*SPACE ⊆ N*X*SPACE
B. *X*SPACE = N*X*SPACE
C. *X*SPACE ⊇ N*X*SPACE
D. None of the above



## Effects of Nondeterminism

Observation so far
- $L$(TM) = $L$(NTM)
- $L$(DFA) = $L$(NFA)
- $L$(DPDA) ⊆ $L$(PDA)   PDAs are nondeterministic
- P ⊆ NP
  - Probably P ⊂ NP (i.e., P ≠ NP)



## Savitch's Theorem

SPACE($f(n)$): Space-bounded by $O(f(n))$

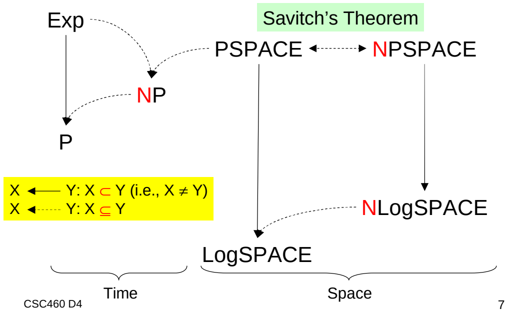
- Given a function $f(n) \geq n$,

  NSPACE($f(n)$) ⊆ SPACE($f^2(n)$)
  - Corollary: NPSPACE = PSPACE
- Proof idea
  - Elimination of exponential factor: Simulate nondeterminism using a stack à la "iterative deepening"   May still take more than P time.
  - Source of the square: Multiplication by the stack height

Why the requirement: $f(n) \geq n$?

Rev. 2

## Complexity Hierarchy






## Group Exercise: Question 3

- *X* vs. N*Log(X)*SPACE
  - E.g., Exp vs. (N)PSPACE, P vs. NLogSPACE
- Which would be correct? Why?

A. $X \subseteq$ N*Log(X)*SPACE
B. $X =$ N*Log(X)*SPACE
C. $X \supseteq$ N*Log(X)*SPACE
D. None of the above

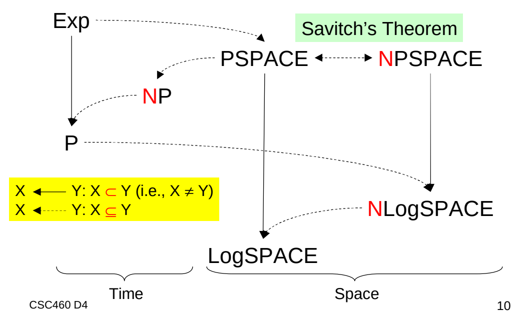


## *X* ⊇ N*Log(X)*SPACE

- Exp ⊇ PSPACE idea
  - A TM using $n$ space can have at most $n \times 2^{O(n)}$ different configurations (~ states, head position, tape symbols).
  - PSPACE would use Exp time.
- P ⊇ NLogSPACE idea
  - MAZE is NLogSPACE-complete (i.e., can simulate any NLogSPACE problem).
  - Analogous to the above case



## Complexity Hierarchy






## D3 Unit Summary

- Time and space complexities have some connection.
- The effect of nondeterminism differ in each case.
- Applications of complexity theory
  - NPC and beyond ⇒ Use of approximation algorithms
  - Effect of limited resources (of different types)



## Unit D4: Overview

- Analyze the effects of parallel processing on theoretical aspects
- Explore examples of parallel processing
- Preview Exercise D4 "Reading: Critical View about TMs"

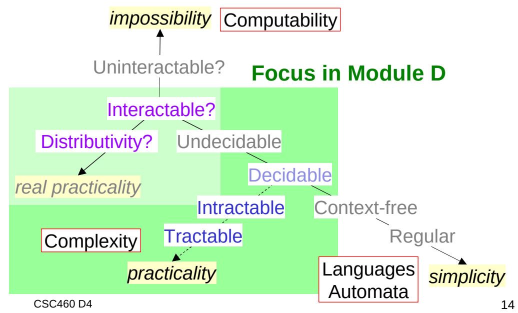
Parallelism as the ultimate dimension in computing?

# Parallel Processing

- Classification
  - A few processors in a single computer
  - Parallel machine
  - Distributed computing [multiple units/locations]
- Questions
  - Decide "undecidable" problems?
  - Move down Chomsky hierarchy?
  - Reduce time complexity (asymptotically)?



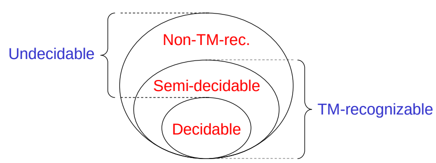
# Overview: Theory of Computation






Review

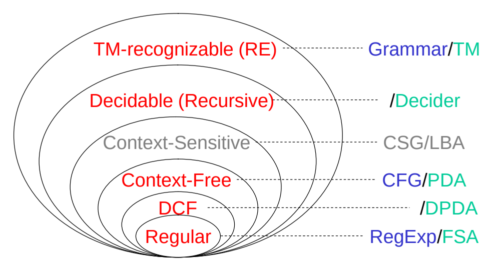
# Computability Summary






Review

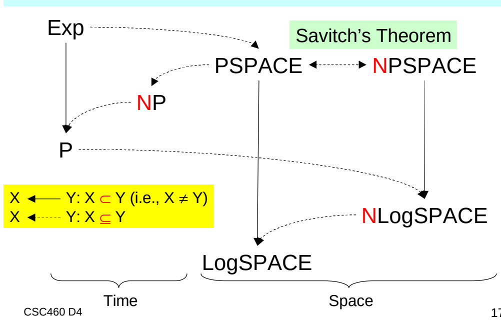
# Extended Chomsky Hierarchy






Review

# Complexity Summary






# Group Exercise 1

A. Choose at least two different types of parallel processing examples
B. Could they decide "undecidable" problems?
C. Could they move down the Chomsky hierarchy?
D. Could they reduce time complexity?

## Classification of Tasks

- Instruction
  - Parallelizable vs. inherently sequential
- Data
  - Disjoint vs. shared



## Classification of Parallelism

- SISD: Single Instruction Single Data
- SIMD: Single Instruction Multiple Data
- MISD: Multiple Instruction Single Data
- MIMD: Multiple Instruction Multiple Data

Flynn [1966]



## Degree of Parallelism

X Inst. Y Data
- SISD
- SIMD
- MISD
- MIMD

May consider different types

- Constant, i.e., a fixed number $k$ of parallelism
- Function of the input size, $n$
  - Assumption: A large number of processors are available so that given any reasonable $n$, every $c$ input units can be assigned a processor (for linear case, other functions possible).

Massively parallel processors



## Group Exercise 2

- Analyze whether the following cases of parallelism would affect: (a) computability, (b) complexity, (c) Chomsky hierarchy.

Case 1: Constant factor

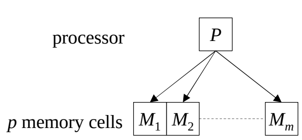
Case 2: Function of the input size, $n$
  - Assumption: A large number of processors are available so that given any reasonable $n$, every $c$ input units can be assigned a processor (for linear case, other functions possible).

Modeling parallelism?



## RAM

- Random Access Machine

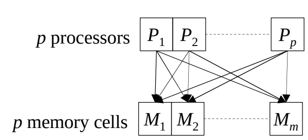



  - Processing cycle: read-compute-write
  - Computation type: arithmetic
  - Cf. URM, which is more primitive



## PRAM

- Parallel Random Access Machine




  - Processing cycle: read-compute-write in sync
  - Processors run the same program, possibly different interpretation based on their index

## Models of Parallel Processing

- PRAM
- Hypercube ($2^d$ processors for $d$ dimensions)
- Bounded Degree Network (at most $d$ connections per processor)



## Example Problems

- Search
- Finding the max
- Summation
- Matrix multiplication $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \quad \text{for } 1 \le i, j \le n$
- Merging
- Merge sort

Handling conflicts
[C: common, E: exclusive
- EREW [not realistic]
- CREW
- CRCW
  - Common write
  - Priority write



## Group Exercise 3

- Find the time complexity and processor complexity (number of processors) for the following problems:

1. Summation
2. Matrix multiplication $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \quad \text{for } 1 \le i, j \le n$
3. Merge sort



## Class NC "Nick's class"

- Informally, efficiently parallelizable class of problems
- Definition: Solvable on a PRAM in log time with polynomially-many processors
- NC $\subseteq$ LogSPACE $\subseteq$ P

Meaning of this?
Example of P problem not in NC?



## Unit Summary

- Parallel processing can affect complexity, and Chomsky hierarchy, but not computability.
  – Some (not all) common problems have exponential speed up with parallel processing.
- The complexity of distributed computing in general is more difficult to analysis.



## Summary Question

- If parallel processing does not affect the computability results, what would do so? Speculate.
- Questions/Comments/Suggestions

Name: ______________________

# Exercise D4, 4/15/05

## Reading: MacLennan

While we have been studying the traditional, TM-based Theory of Computation as a useful tool for analyzing practical problems, we also encountered its limitations at various points. In order to broaden our view, we will read a paper by B.J. MacLennan, who is very critical about TM-based models of computation.

Reference: MacLennan, B. J. 2003. Transcending Turing Computability. *Minds and Machines* 13(1):3-22. [also see others in the course reference list; MacLennan's web site: http://www.cs.utk.edu/~mclennan/]

**Task 1**: Read the paper carefully. Try to relate the author's argument with what you learned in this course.

**Task 2**: Concisely write up your response to the following questions. In doing so, try to apply your analysis to your own mini research paper.

1. What is the main research question of the paper?
2. What is the significance of the research question?
3. Does the paper respond to the research question well?
4. Is the paper organized well?
5. Summarize the author's criticism of the traditional Theory of Computation *in the order of importance* (as you understand).
6. Try to criticize the author's position/arguments. If you cannot do this, explain why.
7. How would the author's criticism about the traditional Theory of Computation affect your analyses of your own problems and/or your choice of the mini research problems that have been done so far, *focusing on computability and (time) complexity*?
8. [Optional] What would the author mean by "pitfalls of learning"? (in the middle of p. 19)
9. [Optional] Discuss whether you would analyze yourself in terms of a Turing machine or a natural computation, referring to the arguments in the paper and relevant properties of yours.
10. [Optional] The author does not directly discuss one subarea, formal languages/automata (except the TM). Speculate what the author would say about the Chomsky hierarchy.
11. [Optional] Speculate on how to define the "power" of natural computation. (in the middle of p. 19)

Be prepared to *discuss* your response in class (no need to prepare for a formal presentation).

Survey: Time spent between classes: ______________

Supplemental notes for the reading

- Effective (p. 4): Roughly, well-defined, step-by-step (e.g., algorithm = effective procedure that terminates)

- Calculus (p. 4): In logic, calculus refers to basically symbol manipulation (i.e., syntax, which is contrasted with semantics or meaning).
- Phenomenological (p. 5): One of the main methods of the phenomenological investigation is to suspend presuppositions (Stewart and Mickunas, 1990).
- Schema (sg.)/schemata (pl.) (p. 6): As an example, $X + Y = Y + X$ is a rule schema, which can be instantiated with various values for $X$ and $Y$.
- Semantic vs. pragmatic effect (p. 8): While semantics normally refers to context-independent part of the meaning of a statement, pragmatics is about how to use symbols in context.
- Generative grammar (p. 8): Basically the same as our use of "grammar" in this course. That is, a grammar is supposed to accept/generate strings.
- Competence/performance (p. 8): These were introduced by Chomsky, and are widely used. Analogous (but not identical) contrasts have been noted by other people as well: e.g., langue/parole by Saussure, I-language/E-language by late Chomsky.
- Tarski (p. 8): The main figure who proposed the "truth-conditional" approach to interpretation of logical statements.
- Löwenheim-Skolem Paradox (p. 10): Here is an example used in my section of Discrete Structures. Mathematicians try to specify the set of natural numbers using a set of statements in first-order logic. However, when they feel they are successful, the specification also specifies unintended structures as well. That is, all the structures of the form NZ*, where N represents natural numbers, Z represents integers, and '*' represents the Kleene closure, i.e., any number of integers attached after natural numbers can still satisfy the specification. A result like this can only be obtained if we carefully relate the syntax (form) and the semantics (content) of a logical system (as done in model theory, part of formal logic). This suggests that there may be a lot of loose ends to formal specification.
- Satisfice (p. 12): "To accept a choice or judgment as one that is good enough, one that satisfies." Herbert Simon is a Nobel laureate (Economics), who is also considered as a pioneer in AI.
- Foreground/background (p. 14): Cf. people's interpretation of the "vase-face illusion."
- Emergent phenomenon (p. 16): One of the main properties of a complex system. Such a phenomenon, which is often unpredictable, may emerge as a result of the interaction among (possibly relatively simple) components.
- Pragmatics/semantics/syntax (p. 17): Pragmatics deals with how language (or other means) is *used* in context. Semantics primarily deals with a *context-independent part of the meaning* of a sentence. Syntax deals with the (recursive) *structure* of a sentence.
- Nekker cube (p. 17): Best to see the image (do a web search). The switching of your perception can be analyzed in terms of "catastrophe theory" (a non-technical intro in, e.g., Gottman, et al. "Mathematics of Marriage"). This type of observation is often used in support of "holistic" position to cognition, cf. "reductionistic" or "analytical" position.

// End

Unit D4 (cont'd)

## Example Problems

- Search
- Finding the max
- Summation
- Matrix multiplication $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} \quad \text{for } 1 \le i, j \le n$
- Merging
- Merge sort

Handling conflicts
[C: common, E: exclusive
- EREW [not realistic]
- CREW
- CRCW
  - Common write
  - Priority write



## Class NC "Nick's class"

- Informally, efficiently parallelizable class of problems
- Definition: Solvable on a PRAM in log time with polynomially-many processors
- NC $\subseteq$ LogSPACE $\subseteq$ P
- Examples?

Meaning of this?
Example of P problem not in NC?



## D4 Unit Summary

- Parallel processing can affect complexity, and Chomsky hierarchy, but not computability.
  - Some (not all) common problems have exponential speed up with parallel processing.
- The complexity of distributed computing in general is more difficult to analysis.



## Unit D5: Overview

- What's wrong with TM or TM-based Theory of Computation?
  - Reading: MacLennan
  - More examples
  - Analyzing the source of the problem
- Preview Exercise D5 "Reading: Super-Turing Thesis"



## Question

- Is the Theory of Computation relevant to all the real-world computational problems?
- Why or why not?



## Group Exercise 1

- Compare your exercises (reading MacLennan)
- Prepare to present interesting points (possibly including similarities and differences)

## Reading: MacLennan (1)

1. What is the main research question of the paper?
2. What is the significance of the research question?
3. Does the paper respond to the research question well?
4. Is the paper organized well?
5. Summarize the author's criticism of the traditional Theory of Computation *in the order of importance* (as you understand).
6. Try to criticize the author's position/arguments. If you cannot do this, explain why.
7. How would the author's criticism about the traditional Theory of Computation affect your analyses of your own problems and/or your choice of the mini research problems that have been done so far, *focusing on computability and (time) complexity*?



## Reading: MacLennan (2)

8. [Optional] What would the author mean by "pitfalls of learning"? (in the middle of p. 19)
9. [Optional] Discuss whether you would analyze yourself in terms of a Turing machine or a natural computation, referring to the arguments in the paper and relevant properties of yours.
10. [Optional] The author does not directly discuss one subarea, formal languages/automata (except the TM). Speculate what the author would say about the Chomsky hierarchy.
11. [Optional] Speculate on how to define the "power" of natural computation. (in the middle of p. 19)



## Where is This Community?

**Social classes**

- Within each colony, there is a strict division of labor. Some are warriors who defend their colony. The farmers collect leaves, and the domestic workers tend the mushroom beds and the young.

**Job description**

- Farmers collect leaves. Well-worn trails are made as they travel to and from their colony. They are selective about the leaves they collect, and will often travel a long distance to find a certain plant. In this way, they spread out their consumption so that trees do not become stripped.
- In special chambers, domestic workers process the leaves into a pulp, making a bed of fertilizer upon which mushrooms are grown. They carefully weed the mushroom bed, ensuring that only one kind of mushroom is grown, and they continually add additional leaves to enrich the crop.



## Swarm Intelligence

- Are ants intelligent?
- Is intelligence in the brain?



## Where is This Community?

- The entire community consists of the near-blind, due to some genetic problem.
- This community survived many years, even against the attacks of invaders.



## Army Ants

- Successful for millions of years.
- How could they function?
  - E.g., how could they find food?

## Swarm Intelligence

- Each ant is not very intelligent.
- Their colonies exhibit complexity approaching that of intelligence.
- The intelligence is in the colonies, possibly including the environment.
  - Cf. the shortest path recorded in the environment (not known by each ant)

cf. Chinese room (Searle)



cf. Mass Stupidity

## Group Exercise 2

- The Irishmen in the area (say, 100 of them) like to go to the local pub when Irish music is played (Thursday evenings) only if the pub is not crowded (say, up to 60 people).

How could they predict the pub is not crowded?



## El Farol Problem

- Example attempts
  - Same as the last week
  - Average of the last four weeks
  - Trend in the last eight weeks
- If they can all predict, all of them would go and the pub will be crowded. False prediction
- There is no absolute way to predict the crowdedness of the pub. Implications?



## Can a TM design this?

AI problems?



## Challenges in/around CS

- Operating System
  - Interprocess communication, I/O, memory
- Software Engineering
  - Specification, verification Doomed to failure?
- AI
  - Games, robotics, NLP, Cyc, 5th Generation AI
- Cognitive Science
  - Theory of mind (cf. Fodor: mind as TM)



## Group Exercise 3

- What would be the essential properties that might make TM inappropriate for modeling the real-world (or computation in general)?

## Observation

- The environment (context) changes over time.
  - No control over adversaries.
  - Future environment is unknown/unpredictable.
- The environment may affect agents (including their thinking process) through interaction.
  - Nonmonotonic (e.g., belief change)
- TMs cannot handle the passage of time.
- TM learning is not impossible, but impractical.
- Tentative conclusion: Interaction is non-TM-recognizable.



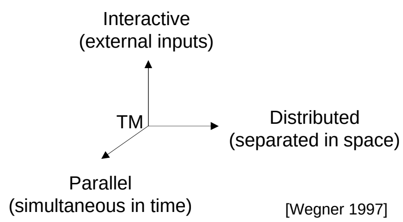
## Algorithm vs. Interaction

| Algorithm | Interaction |
|---|---|
| Closed-book exam | Open-book exam |
| Sales contracts | Marriage contracts |
| Rationalism | Empiricism |
| Compositional | Non-compositional |
| Reductionistic | Holistic |
| Monotonic | Non-monotonic |
| TM | must be something... |

We find parallel algorithms and distributed algorithms, but no interactive algorithms.



## Elements of Modern Computing




[Wegner 1997]

Effects on "Theory"?



## Paradigm Shift

- Astronomy: Ptolemaic → Copernican
- Physics: Classic → Relativistic
- Mathematics:
  - Hilbert's program → Gödel's incompleteness
- Linguistics
  - Transformational → Mildly Context-Sensitive
- Psychology
  - Behaviorism/cognitivism → Social/cultural

Computer Science?



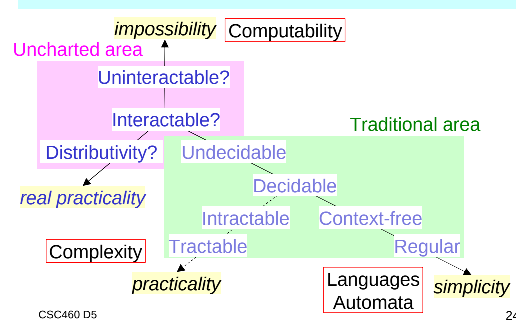
## Computer Science

- "goto" considered harmful: Structured programming
- "assignment" considered harmful: Functional programming
- "procedure" considered harmful: Logic programming, object-oriented programming
- Too much of the TM-based "theory" considered harmful: model of interaction

Turing and von Neumann hinted models beyond TM.



## Extended Theory of Computation



## Unit Summary

- TMs are too limited as a model of "real" computation.
  - Call for a more realistic model
- The future of Computer Science may depend on a paradigm shift from the current TM-based theory to a new theory of interaction.
  - The current "Theory of Computation" hopefully provides useful information for this transformation.
- It might even be the theory of "everything," cf. Unified Theory in physics. But feasible?



## Summary Question

- What would a more realistic model of computation look like? Speculate.
- Questions/Comments/Suggestions

References available on-line

Name: ______________________

# Exercise D5, 4/19/05

## Reading: van Leeuwen & Wiedermann

After reading and discussing MacLennan, we may or may not agree with all of his points. However, one point we cannot overlook is that there is a growing number of researchers who take a position similar/related to MacLennan (see the course reference list for more information; Copeland [2003] is a detailed review). In order to gain additional insight into the area beyond TMs, we will read a paper by Jan van Leeuwen and Jirí Wiedermann, who propose a "super-Turing" thesis to capture the notion of computation at a higher level, potentially more applicable to modern computing.

Reference: van Leeuwen, Jan and Wiedermann, Jirí. 2001. The Turing machine paradigm in contemporary computing. In *Mathematics Unlimited - 2001 and Beyond*, eds. B. Enquist and W. Schmid, 1139-1155. Springer. [the distributed copy, available on-line, is a draft version of (i.e., not necessarily identical to) the listed paper; van Leeuwen's web site: http://www.cs.uu.nl/people/jan/]

**Task 1**: Read the paper carefully. Try to relate the author's argument with what you learned in this course. Note: You may skip the details of certain technical parts, e.g., the "advice" function.

**Task 2**: Concisely write up your response to the following questions
1. What is the main research question of the paper?
2. What is the significance of the research question?
3. Does the paper respond to the research question well?
4. Is the paper organized well?
5. Summarize the author's arguments *in the order of importance* (as you understand).
6. Were you convinced by Proposition 5 and its proof (p. 10)?
7. What is the difference between the (traditional) Church-Turing thesis and the extended Church-Turing thesis (proposed in the paper)? What kind of impacts could the extension have?
8. Try to criticize the author's position/arguments. If you cannot do this, explain why.
9. How would the author's arguments affect your analyses of your own problems and/or your choice of the mini research problems that have been done so far, *focusing on computability and (time) complexity*?

Be prepared to discuss your response in class.

Survey: Time spent between classes: ______________

Notes for the reading
- Oracle computation (p. 4): If a TM is given an oracle that decides some undecidable problem (set), then the TM could recognize a non-TM-recognizable language. Naturally, the increased power in this case comes from the power of such an oracle, which is beyond a TM.
- Advice (p. 6): A limited version of oracle (because an oracle could be excessively powerful). Discussed more in detail in Section 3.2.1.

- Mapping $\Phi$: $(\Sigma^k)^\infty \rightarrow (\Sigma^l)^\infty$ (p. 7): A function whose input is an infinite sequence of $k$ symbols on $k$ ports and whose output is an infinite sequence of $l$ symbols on $l$ ports. As an example, suppose a machine with a keyboard and mouse as input device and display and printer as output device. Imagine that the keyboard deals with symbols from the set $K$; analogous for other devices: $M$ for mouse, $D$ for display, and $P$ for printer. Then, the mapping/function would be as follows: $\Phi$: $(K \times M)_{\text{[for time 1]}} \times (K \times M)_{\text{[for time 2]}} \times \ldots \rightarrow (D \times P)_{\text{[for time 1]}} \times (D \times P)_{\text{[for time 2]}} \times \ldots$
- Advice function (p. 8): The authors consider that oracle computation without limit is too powerful (p. 4). What they propose is to use a limited version of oracle called "advice." Their "advice" takes the form of function: given a natural number, the function returns a string. The power of the function is basically "polynomial," but include certain "randomization" possibilities, useful for reflecting unknown future. Here, we consider this function as a "reasonable" way to integrate mostly tractable but not completely predictable. Some additional notes are included below (indented).
  - Bounded by $S(n)$ (p. 8): $S(n)$ refers to "space bound" with respect to the input size (cf. $T(n)$ for "time bound). Note that this bound applies to the F part of the class specification "C/F" (see below).
  - Class C/F (p. 8): The main motivation for this definition is to introduce the class *P*/*poly* (see below). Skip the details.
  - Class *EXPTIME* (p. 8): This corresponds to the class Exp used in this course.
  - Class *poly* (p. 8): The class of polynomially (size-)bounded advice function. Recall that the space bound $S(n)$ applies to F, and *poly* is one case of F.
  - Class *P*/*poly* (p. 8): Roughly, this class is a "non-uniform" (evolutionary, upgradable, etc.) analogue of *P*. It contains *P* (i.e., $P \subset P/poly$) as well as some undecidable sets (due to its upgradability, which is not fixed at the beginning). The comparison with *NP* is not yet known. The fact that *P*/*poly* contains *RP* and *BPP* (see below) suggests that *P*/*poly* captures some randomized behavior.
  - Class *NP*/*poly* (p. 8): Ignore this class.
  - Class *RP* [Randomized Polynomial Time] (p. 8): A language is in *RP* if there is a boolean verifier computable in deterministic polynomial time (or finding a boolean satisfiability answer with a NTM) such that the input is accepted at least 1/2 of the time.
  - Class *BPP* [Bounded-Error Probabilistic Polynomial Time] (p. 8): Analogous (or more confident version) to RP, except that the input must be accepted at least 3/4 of the time (the value could be others, e.g., 2/3, as long as it is strictly greater than 1/2, as they can approach 1 by repeated multiplications).
- $\omega$-automata (p. 9): Variants of FSAs that accept infinite strings (not just potentially infinite as in the case of FSAs).
- Non-recursive (p. 10): Beyond recursive, i.e., r.e. (TM-recognizable) or non-r.e. (non-TM-recognizable).
- $\langle M \rangle$ (p. 10): A standard way of indicating that this is a representation of a TM. We could consider this as the TM's formal definition, as discussed in class.
- Proof of Proposition 5 (p. 10): The basic idea is that the advice function identifies the maximum number of steps for a TM whose representation is of some length. Although the advice function adds power to the TM, it is still within the bound. So, it is not considered unreasonably powerful. However, we know that such a function is not decidable by a standard TM.
- Theorem 6 (p. 11): Ignore it.
- S = ($\gamma$) (p. 11): Here the site machine is defined (as a 1-tuple) in terms of the function $\gamma$ from time to a hardware/software description (p. 7).
- Instantaneous description (p. 11): A snapshot of a machine. The instantaneous description of a TM (or similar machine) includes the current state, head position, and the tape symbols. Note that if $n$ tape positions are used, there would be at most $|\Gamma|^n$ symbol possibilities, where $\Gamma$ is the set of tape symbols; this suggests that it would have taken at most exponential time to have used the space.
- Infinite circuit families (p. 16): Ignore this.
- BSS model (p. 16): After Blum, Shub, and Smale. The problem of deciding general existential formulas in the first-order theory over the reals, which is a real analogue of NP-complete.
- Neuroidal networks (p. 16): A specific version of a neural networks, proposed by a computational learning expert Valiant.

// End

## Unit D6: Overview

- Explore models of super-Turing computability
  - Reading: van Leeuwen & Wiedermann
  - Other models
- Summaries: Unit, Course
- Exercise D6 (Module D Comprehensive) "Mini Research"



## Group Exercise 1

- Compare your exercises (reading van Leeuwen & Wiedermann)
- Prepare to present interesting points (possibly including similarities and differences)



## Reading: van Leeuwen & Wiedermann

1. What is the main research question of the paper?
2. What is the significance of the research question?
3. Does the paper respond to the research question well?
4. Is the paper organized well?
5. Summarize the author's arguments *in the order of importance* (as you understand).
6. Were you convinced by Proposition 5 and its proof (p. 10)?
7. What is the difference between the (traditional) Church-Turing thesis and the extended Church-Turing thesis (proposed in the paper)? What kind of impacts could the extension have?
8. Try to criticize the author's position/arguments. If you cannot do this, explain why.
9. How would the author's arguments affect your analyses of your own problems and/or your choice of the mini research problems that have been done so far, *focusing on computability and (time) complexity*?



## Sample Problems

- Driving from one place to another
- "Real" web programming (arachnidan)
- Bird flocking
- Psycho-social behaviors
  - E.g., child-parent attachment, language
- "Gaia" hypothesis (the entire earth as an organism)
- The universe

cf. Chomsky

Can site machines/TM with advice do these?



## Super-Turing: Requirements

- Interactivity: Can accept input and deliver output in the middle of computation
- Robustness: Can deal with fuzzy, unknown, unpredictable environments (inputs)
- Adaptability: Can change according to the environment (i.e., learning)



## Super-Turing: Assumptions

- Extension: The TM-based "theory" is to be a special case of the theory of interaction.
- Multiple levels of representation (cf. Church-Turing Thesis)
  - Machine models (cf. TMs, URMs, RAMs)
  - Mathematical/functional models (cf. recursive functions, $\lambda$-calculus)
- Ockham's Razor: Choose the simplest if multiple options are available

Why this many?

## Super$^{TM}$ Models

- Site machines (concrete machine model) [van Leeuwen & Wiedermann]
- TMs with advice function (abstract machine model) [van Leeuwen & Wiedermann]
- Interaction machines (persistent TMs) (abstract machine model) [Wegner & Goldin]
- yet another machine model (intentionally kept untold at this point)
- $\pi$-calculus (functional model) [Milner and others]
- Neural models [e.g., Siegelmann]
- Quantum/relativistic computing

cf. on-line references



## Interaction Machine (IM)

- Note: TM computation = Single session
- IM computation = Multiple sessions
- Interactivity: Realized as *repeated* TM sessions where information can be *stored* on a working tape
- Multiple input streams on multiple tapes
- An extension of a multiple-tape TM

Variations?



## Variations

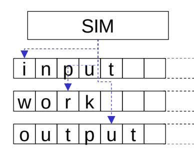
- General: MIM (Multiple-stream IM)
  - Model of distributed computing
- Special: SIM (Single-stream or Sequential IM)
  - Model of 2-agent interaction

MIM: not yet well-developed



## Single-stream IM (SIM)

- Single input stream on a single tape
- Work tape will be preserved (persistent).
- Behavior: set of I/O streams (cf. TM behavior: set of I/O pairs)




Example: Answering machine
(input, work) => (output, work)
- (record Y, X) => (ok, XY)
- (playback, X) => (X, X)
- (erase, X) => (done, ε)



## Properties of SIMs

- SIM states depend on the content of the working tape [no bound].
- Infinitely many states (cf. finite TM states)
- Uncountably many SIM's (cf. countably many TMs)
- Can handle uncountable sets (cf. TM can handle countable sets, i.e., uncountable ~ unsolvable)



## Properties of MIMs

- No single observer (on a single stream) has complete information.
  - Cf. A SIM has complete information (about the interaction between the two parties).
  - Analogy: "Blind men and an elephant"
- Multiple streams cannot be serialized.

Example: Delegation
- Stream 1: (main-in, main-out)
- Stream 2: (sub-out, sub-in)
- Coordination: (main-in, sub-out, sub-in, main-out)

# Comparison

- TM: 1-agent systems ⇔ countable sets
  - Traditional Church-Turing Thesis for algorithmic computability (TM, λ-calculus)
- SIM: 2-agent systems ⇔ uncountable sets
  - Extended Church-Turing Thesis for sequential interaction (SIM, calculus?)
- MIM: *n*-agent systems ⇔ ?
  - Ultimate Church-Turing Thesis for general interaction (MIM, calculus?)

TM as amnesic SIM



# Analysis of IM

- Interactivity
- Robustness
- Adaptability



# Functional Model

- TM ⇔ λ-calculus
- SIM ⇔ ?
- MIM ⇔ ?



# λ-calculus

Formal representation of functions

- Abstraction: $\lambda x.exp$ [define a function]
- Application: $exp_1\ exp_2$ [give an argument]
- β-reduction: $(\lambda x.exp)\ y$ [use a function]
  - The result is $exp$ where the instances of $x$ is replaced with $y$.
  - E.g., $(\lambda x.f(f(x)))\ 0 = f(f(0))$

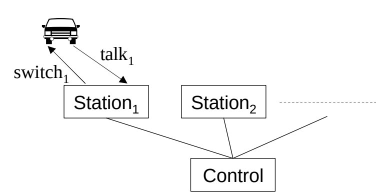


# λ-calculus Examples

- Identity function: $\lambda x.x$
- Church numeral
  - Representation of a natural number $n$: $\lambda f.\lambda x.f\ n(x)$
  - $0 = \lambda f.\lambda x.x$ [Note: $\lambda f.\lambda x.f\ 0(x)$]
  - Succ $= \lambda n.\lambda f.\lambda x.(f\ ((n\ f)\ x))$
  - $1 =$ Succ $0 = \lambda f.\lambda x.(f\ (((\lambda f.\lambda x.x)\ f)\ x)) = \lambda f.\lambda x.(f\ (\lambda x.x\ x))$ $= \lambda f.\lambda x.(f\ x)$ [Note: $\lambda f.\lambda x.f\ 1(x)$]
- Conditional
  - $T = \lambda x \lambda y.x$ [i.e., picking the first argument]
  - $F = \lambda x \lambda y.y$ [i.e., picking the second argument]
  - B M N (if B then M else N) where B is T or F



# Example Scenario



## π-Calculus

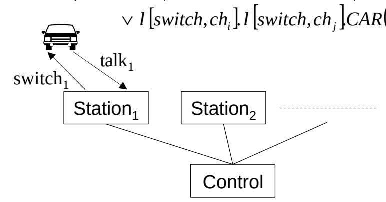
Changes to λ-calculus

- Output mechanism
  - Cf. β-reduction: $(\lambda x.exp)\ y$ as an input mechanism
- Parallel composition of processes
  - Commutative, associative
- Meaning as state change
- Referencing (by name)
  - Cf. pointers in modern programming languages



## π-Calculus Example

$$CAR(talk, switch) \stackrel{def}{\equiv} out\,[talk, word].CAR(talk, switch) \vee l\,[switch, ch_i].l\,[switch, ch_j].CAR(ch_i, ch_j)$$




$$NETWORK \stackrel{def}{\equiv} \underbrace{(CAR(talk_1, switch_1) \mid STATION_1 \mid \ldots \mid CONTROL)}_{talk_1, switch_1}$$

simplified notation



## Analysis of π-Calculus

- Interactivity
- Robustness
- Adaptability



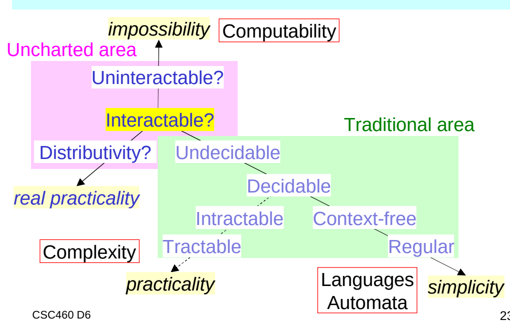
## Super-Turing Thesis

- Would all models of interaction converge?
- Would it be the next level of computation/interaction/emergence?

Both Turing and von Neumann have foreseen super-Turing computation, even though their names are more strongly associated with algorithmic computation. Why did most computer scientists pick up only the algorithmic aspect of their ideas?



## Extended Theory of Computation






## Unit Summary [critical view]

- TM-based theory
  - Stimulus-response without thinking
  - Vertebrate with a disabled brain
    Cf. "chicken without a head"
- Theory must evolve into a more sophisticated beast.
  - Modeling full-fledged interaction
- Some building blocks are emerging.

Preview

## Module D Evaluation Workshop

- Mini research presentations
  - Know the time limit; Think big; Be creative
- Peer discussions
- Reflection and self-evaluation

Practicum evaluation
- Wed., Apr. 27
- Detailed instructions will be distributed later (but already available on-line)



## Course Summary

- Understanding the "Theory" $\Rightarrow$ Computer scientist (good at solving algorithmic problems)
- Understanding the limitations of the "Theory" $\Rightarrow$ Philosopher (good at solving "real" problems)
- Change in your brain?

Sin minas, no hay tango.
Sin teoría, no hay ciencia.

Surveys

Name: ______________________

# Exercise D6 (Module D Comprehensive Exercise), 4/22/05

Note: This exercise is announced in advance so that we can think about it through this module. The due date of this exercise is the date of Module D Evaluation Workshop.

## Part 1: Mini Research

One of the main goals of this course is to be able to see real-world problems in a principled way assisted by the concepts and tools in the Theory of Computation. To practice making such a connection between problems (concrete) and the Theory (abstract), you will spend part of your out-of-class time to conduct mini research. You will identify your own research question which is related to a practical problem of your interest, and respond to it using the skills gained in this (and possibly other) courses. Your research question can be *related* to your initial problem and/or earlier mini research questions, or can be an entirely new one. If you choose a related question, avoid repeating what you have already found and/or written; that is, if you continue on an earlier topic, you must have *something new* (and must explicitly state what is new). If you want to work on a borderline/questionable case, consult the instructor. Some parts of module D take-home exercises will guide you through the process of completing this comprehensive exercise: e.g., Exercise D2 asks for the choice of your research question and Exercise D3 asks for an outline of the paper.

**Task 1**: Write a mini research paper. In your paper, address the following points:

- Clearly identify your research question early in the paper (if necessary, provide some background information to introduce the question)
- Clearly identify the cost/significance of the research question (this can be done by connecting your research question with a practical problem)
- Organize your paper so that the paper responds to the research question *applying the concepts/techniques discussed in this course*, at a level reasonable for the available time and your capacity
    - It would be ideal if you can refer to all three subareas of the traditional Theory of Computation. If this is not possible, try to include as many Theory topics as possible.
    - While it would be great if you can offer a "solution" to your research question, you should be content if you describe your attempt to solve it.
- Conclude the paper with a punch-line statement of your response (and if you were unable to offer a solution, state the information necessary to solve or complete your research)
- Consider to have multiple sections (Introduction, middle section(s), and Conclusion). However, there are no other formatting or length requirements.

Note: Recall that paper writing (as well as coding) can be seen as a context-free activity (if you cannot remember, make sure to understand this by discussing with other students or the instructor). When you organize your paper, try to apply this concept.

**Task 2**: Prepare for a five-minute, informal presentation (to take place during Module D Evaluation Workshop). You can (but not required to) use PowerPoint slides (no more than several slides) if the file is accessible without logging into your Novell account (e.g., stored in

your public HTML folder on a floppy diskette). Attach to this exercise the printout of your presentation materials, if any (preferably, in the format with multiple slides per page).

### List of sample research questions from Exercise A6/B0

1. Can **organizational dynamics** be modeled as an algorithm?
2. Can **evolution** be modeled as an algorithm?
3. Can **ecology** be modeled as an algorithm?
4. Can **human development** be modeled as a computer?
5. Can our **minds** be modeled as a computer?
6. Can **vision** be modeled as an algorithm?
7. Can **learning** be modeled as an algorithm?
8. What would be the minimal mechanism to process **human language**?
9. Can the entire situation of an arbitrary **game** be modeled as an algorithm?
10. Would “perfect” **user modeling**, e.g., for web search, be possible?
11. Would “perfect” **computer security** be possible?
12. Can the entire process of **software engineering** be modeled computationally?
13. Can **computer networks** be modeled as a single computer?
14. Could **biology** be reduced to physics?
15. Could some computer generate real (not pseudo) **random numbers**?
16. Would it be possible to decide whether the given numbers are **random**?
17. Would **randomization** affect computability and/or complexity?
18. Would **parallelism** affect computability and/or complexity?
19. Would **artificial neural network** be more powerful than TMs?
20. Would **cellular automata** be more powerful than TMs?
21. Would the use of **analog** (or fuzzy) values affect computability?
22. Would relativistic, quantum, or some other **modern-physics**-based computation surpass TMs?
23. Can all the cases of **on-line algorithm**s be simulated by off-line computation? [On-line algorithms would obtain inputs as the time progress. Off-line computation would provide all the possibilities as input at once. Cf. function-to-relation conversion used to fold the output within the input]
24. Would **oracle computing** affect computability? [Oracle computing: A TM with the capability to ask questions to another mechanism]
25. Would **persistent TM** be able to compute more than the standard TM? [Persistent TM: Multiple sessions of TM operation with some memory between them]
26. Would **accelerating** a TM give more power?
27. Would slight **error tolerance** affect any aspect of the Theory of Computation?
28. What would be the ability of a finite automaton with a **queue**?
29. What would be the effect of “**constant**” (as in complexity analysis) in practice?
30. Can any **mathematical function** be represented computationally?
31. What exactly are **power sets** doing to the Theory of Computation?
32. Is what you can do in **logic** the same as what you can do with computation?
33. If you have a research question of **your own**, please consult the instructor first.

## Part 2: Evaluation Form and Supporting Notes

Review the evaluation procedure. Then, complete your evaluation form and supporting notes. Bring them to the evaluation workshop (hard copy). Print them well in advance so that you can avoid potential problems, e.g., not being able to print just before the evaluation.

Survey: Time spent between after Unit D6 before the evaluation workshop: ______________

// End

Name: ______________________

# Practicum Evaluation, 4/27/05

In this course, critical analysis has been one of the main criteria associated with a learning goal. Since you have been through various topics in the theory of computation, you are in a good position to review and give constructive feedback to your peers. So, take this opportunity seriously and try your best to contribute the student body. Although this event takes place after the final evaluation workshop, it still is a part of the course evaluation. The instructor will check the criterion only after receiving this sheet accompanied by your evaluation forms and confirmed that you demonstrated your critical analysis skills. As in other exercises, no grades will be assigned. The instructor may note his comments on your evaluation, if necessary.

Here is the procedure:

1. Identify the presentations you are assigned to evaluate. The information is available on-line (linked from the CS home page) and posted everywhere.
2. Obtain practicum feedback forms and fill in one form per presenter. Unless otherwise required (e.g., by the instructors of other courses), keep your forms anonymous.
3. When you are done, hand in your forms to the instructor attached to this form. DO NOT deposit the forms in any of the appraisal boxes.
    - The instructor will make photocopies of your forms, and will forward the original forms to another instructor or deposit in an appraisal box, depending on the information you supply below.
4. If you are required to evaluate also by other instructors, please list the names of the instructors below. If they have specific requirements, by all means satisfy them.

Instructors who also require evaluation:

______________________________________________________________

______________________________________________________________

______________________________________________________________

// End

**Announcements** (view in a separate page) "); ?> Last modified: " . date( "F d, Y", getlastmod() ); echo ")"; ?>

- See the supplemental notes to Unit D3 on QBF and time-space tradeoff here. [4/13/05]
- A sample response to Ex D1 Part 1 (big-*O* notation via the graphing tool) is available here. [4/8/05]
- Link to an interview with Dean Kamen, the inventor of Segway, on NPR Fresh Air. He emphasized the importance of crossing between concrete and abstract ideas. [4/7/05]
- See the supplemental notes to Unit C6 on "problems" and "languages" here. [4/5/05]
- There are a few additional materials: [3/7/05]
    - Compilation of Ex B5 heuristics
    - Additional notes on Church-Turing thesis and Rice's theorem
    - Extra problems on computability (optional)
- My response to your summary questions in Unit B4 is available here. [2/25/05]
- An example of binary adder is added here. [2/7/05]
- Sample problems from Spring 2003 are availalbe here and also among the top bar selections. [1/28/05]
- A preliminary version of the references for the course is available here. This file will be updated in the future. [1/25/05]
- Welcome to CSC460 course web site! A preliminary version of the syllabus is up now. The final version and more components will be added as they become available. You are welcome to browse the web pages. You can at least see a little more details about the schedule, evaluation, etc. There are certain aspects of this course that are different from many other CS courses. Please find out whether you are willing to do this. By the way, there is nothing you need to do until the first day of class. Furthermore, you'd better wait to buy the textbook, because I will comment on how we will use it. [12/22/04]

CSC460 Theory of Computation (Spring 2005)
Reflective Essay
Nobo Komagata
May 4, 2005

For the past few years, I have been sharing my own reflective essay with students at the end of each course. Here is one for this course; it's a little long. Note that the links in this document are clickable.

**Grades**

Your course grades have been reported to the College; both your Module D and course grades are also available on SOCS (https://socs.tcnj.edu/). I started these notes with "grades" not because it is most important (although some student might think that way) but because it is the least interesting part for me. In my opinion, the letter grade you receive for this course (or possibly other courses as well) means almost nothing. Throughout the semester, we have been evaluating ourselves with a substantial amount of information, which is obviously not reducible to a single letter. So, I hope you know your "real" assessment. I think you all achieved the learning goals as a result of good deal of effort.

**Practicum Evaluation**

First, I would like to thank you for participating the practicum presentations as evaluators. The main reason for this requirement was to promote practicum participation. This stems from some faculty members' experience until a few years ago, when practicum presentations were taking place often with a very small number of audience. Probably, participation as requirement is not a solution. If presentations are sufficiently interesting and informative, there will be audience. In Unit B7, which was optional and was attended by four students, I asked them whether the College would function if there are no requirements. The response was negative. But there are some aspects of life which are not required but done regularly. Are people required to watch TV, eat junk food, marry, etc.? My personal preference is to live in a society where people work not because they are required but for other (hopefully good) reasons.

**Mini research**

The submitted papers are now available on-line (http://www.tcnj.edu/~komagata/csc460/05s/students/). It was quite interesting to see your contributions to the final evaluation workshop, despite the limited time you had for the project. I felt that the workshop appeared as some sort of collective thought or intelligence (cf. global consciousness). It was a kind of forum where the participants were able to share ideas relevant to the course topics, while no single participant was able to predict the progress of the workshop. So, to me, it was a good example of complex system (http://en.wikipedia.org/wiki/Complex_system), which I will occasionally touch on below .

Instead of writing comments on each presentation, let me write a single prose to address the presentation topics. Although the topics of this course is by all means the Theory of Computation, we talked a lot about other things. As we discussed frequently, unless we can relate the course topics to our lives, it is hardly possible to get interested. I think you all did a good job on this point. Sometime last year, you all decided to take this elective course. Was it a

result of “determinism” (i.e., everything is pre-determined) or did you apply your “free will?” Did you do things when they are optional? Were your decisions affected by any of the following: (1) massive parallel processing with large amounts of concurrent data, (2) ability to adapt, and/or (3) some element of randomness (**Scott**)? While most currently used programs would not satisfy all these conditions, if there is one with these three properties, would it behave like us, exercising “free will?” An interactive TM with advice (van Leeuwen & Wiedermann) can satisfy these; advice function can contain some randomness. In a sense, nondeterminism may be broader than free will. Have you seen a movie called “Sliding Doors?” It is a charming film, which illustrates a potentially large effects of a very small difference (cf. “butterfly effect” in chaos theory). This is also related to a source of the failure of symbolic AI, i.e., “ramification problem.” A related but slightly different property is “predictability.” If everything is predictable, we could collect all the future inputs and give them to a TM all at once (“off-line computing”) instead of feeding the information as it becomes available (“on-line computing”). Off-line computing is an active area of research. However, if the advice function of a site machine involve unpredictability, it may not be able to realize everything off-line.

Regardless of our philosophical position, it is certain that our brains work hard to bring about various changes in our bodies. For example, while strolling a shopping mall, we may be using our memory in a way analogous to a PDA (**Marisa**). However, in general, memory appears much more flexible. Even remembering a story would be beyond context-free because we can in general recall any part of the story (cf. **Marisa**). Now, recall that any (standard) TM-based model is “amnesic” in the sense that the result of a single session of computation is not retained. Many super$^{TM}$ models overcome this deficiency by the use of some sort of persistent storage. However, even this modification may not be accounting for human memory. For example, human memory is said to be *constructed* rather than recalled (Engel). What kind of model would explain cases such as anterograde amnesia (**Mike**)? Maybe, we will be able to signal a problematic area of the brain from a PDA through *MemoryGate* (cf. **Jared**).

In practice, though, we are far from such a goal. In general, it is *impossible* to “perfectly” model most of natural phenomena (cf. **Jon**). Natural computation involving “continuity” would face uncountability, beyond TMs (**Jon**). But even countably infinite inputs cannot be placed on the tape of a TM. It would also be impossible to accomplish “perfect” computer security (**Megan**). However, the notion of being “perfect” is difficult to nail down. While a computer scientist might acknowledge “perfect” computer security if someone proves $\mathsf{P} \neq \mathsf{NP}$ (**Megan**), one might say that a “perfectly” secure system must be impossible to crack. Being in $\mathsf{NP}$ simply suggests that it take some time; it certainly accepts that it would be possible to crack (with unrealistically long time). Well, I feel that the demand for remembering all the passwords is way over my computability. In fact, I have a password-protected file that stores passwords, which are again encrypted in my own mnemonic. Honestly, I don’t think this is the future of computer security. In a sense, algorithm-based cryptography is an object-level approach to computer security; in many cases, one needs to deal with security issues at a meta-level. Often we get stuck at the object level and are not seeing the same problem from a meta-level; this situation can be observed in various cases including competitions/games. In a sense, computer security can be characterized as an evolutionary game between computer security developers and hackers. A new ideas in this game might be more evolutionary than algorithmic.

As a tool to capture some aspect of evolution, cellular automata (CAs) are becoming a common tool. While it has been shown that CAs can simulate TMs (Wolfram via **Mark**), the other direction is trickier (cf. **Mark**). Wolfram shows that some segment of CA behavior can be simulated by a TM. However, since CAs would continue indefinitely, unless all the ever-growing patterns can be classified based on a TM computation, it would be premature to say that TMs can simulate CAs. This may be the case, because it has been shown that the bulk of CAs can be reduced to a set of deterministic patterns (Israel & Goldenfeld). But the most striking claim of Wolfram seems to be "Principle of Computational Equivalence," roughly "almost all processes that are not obviously simple can be viewed as computations of equivalent sophistication." To me, this suggests some kind of computation beyond TM. However, it is also true that CAs appears more limited in terms of (external) interaction compared to, say, site machine.

In addition to memory, other types of cognition were brought up. For example, while we humans process vision in real time, it still would be non-TM-recognizable partly due to the involvement of learning (**John**). Another aspect of cognition, common sense knowledge is the focus of a long-standing project called CYC (**Jared**). This project seems to be following the failed track of the Fifth Generation AI, possibly due to its heavy reliance on logic. While first-order logic (FOL) is extremely useful on various fronts, it has its own limitations. When Hilbert proposed that all of mathematics can be formalized in FOL ("Hilbert's program") and Russell tried to complete the program, Gödel showed that a consistent logic that can count cannot prove all true sentences ("Gödel's Incompleteness Theorem") (**Greg**). As mentioned in one of the supplemental notes, this result is directly related to the undecidability of FOL. As for complexity, universal quantifiers are bottleneck (**Greg**), for proving the sentence, corresponding to the burden on the falsifier. But due to the "duality" of the universal and existential quantifiers, disproving a sentence calls for analogous burden on the verifier. So, to be able to say whether a sentence is true or false, both types of quantifier contribute the computation equivalently, leading to exponential growth. Pointing out various problems with FOL, Hintikka proposes a variant of FOL which is more tightly controlled by game-theoretic semantics (Hintikka). Although this idea has not been taken up seriously by traditional mathematicians, it might be wise to listen to such a voice; we might actually observe a paradigm shift in logic, parallel to the Theory of Computation. Note that the syntactic analysis of FOL is a quite different aspect. The backbone of FOL sentences can be "parsed" with a PDA (**Greg**). However, just like modern programming languages, variable coreference requires more than that.

While cognition is more actively studied by psychologists, emotion may hold a key to mind-computer analogy. It is common to think that emotion cannot be "computable" (**Marisa**). I am in general sympathetic to MacLennan's arguments and tend to emphasize the limitations of algorithmic computation. But here, I would like to point out some algorithmic aspects of emotion. Have you ever mistaken a coarsely rolled rope on the ground as a snake and jumped with fear? We can obviously recognize a snake, but it takes time, suggesting that vision is not so cheap (**John**). To avoid potential danger involving a snake, our brains are also equipped with a sort of "short" circuit of emotion that induce a quick action without detailed visual processing (LeDoux). There are many other aspects of emotion that can be reduced neurobiology. For example, it has been studied how infants process visual information and that lead to emotional change in their bodies (e.g., Schore). We also know that an appropriate amount of emotion (e.g.,

frustration, comfort from sharing ideas, surprise from discovery) is essential for learning (note that excessive emotion would lead to traumatic amnesia). If all sorts of emotions can be analyzed and synthesized in the form of robot, it might be possible to generate what we call emotion in a non-organic form. Then, there might be a pathway from a PDA to an anterograde amnesic (**Mike**) or a TM. But until such a day comes, if at all, we have music to appeal to our emotion directly.

As the Church-Turing thesis defines the informal notion of *algorithmic* computation, it would be possible to define the informal notion of enjoyable music (**Eric**). But unlike "algorithms," which are more objective, musical enjoyment is more subjective. Then, would it be really recursive (cf. **Eric**)? For example, our tastes change over time (after listening to a Cuban pianist, Rubén González, I started to feel some other pianists are no longer as good) and are often unpredictable (there are too many musical "strings" which I have never encountered). We might want to respond to such a question by focusing on the domain where TM can decide, or using a super$^{TM}$ model instead.

**This course**

As a part of my reflection, I would like to write a little bit about this course from my view point. A preliminary version of this course was run two years ago as a special topic course. At that time, I was not focusing on complex systems as much as today. As a result, I emphasized the significance of the Theory more than its limitations. That is, I was trying harder to "sell" the Theory. It was not as successful as I expected. During that semester, I was asked to teach IMM220 "Principles of Interactivity." The more I thought about interactivity, the more I felt difficult to sell the traditional Theory. At the end of that semester, I prepared an earlier version of super-Turing lectures, mainly based on materials by Kennedy & Eberhart, Wegner, Goldin, and Milner; I was not aware of MacLennan and van Leeuwen & Wiedermann then. This semester, I approached this course more critically, toning down the significance of the Theory and discussed a lot more about the limitations, as you are well aware of. I am glad to have done this. While I used the traditional Theory a lot during my graduate work (e.g., decidability analysis of pragmatic process, grammars/automata to analyze/process natural language syntax, complexity analysis of parsers), there are so many areas/phenomena that escape the traditional Theory. You know the story.

Although I re-used many slides from the previous run, I revised the course substantially: new learning goals, new exercise problems (and exercise pattern) including new reading, new evaluation procedure, new in-class exercise discussion pattern (designated pairs), etc. Again, I'm glad that I did these. I tried to write more about "why" we do certain things (i.e., significance), e.g., at the beginning of each exercise. I do frequent exercises because I know from the education research literature and my own experience that shorter frequent exercises with timely feedback are more effective than longer infrequent exercises. At the same time, I let you decide on the level of detail so that you can adjust the amount time you use for each exercise. I also tried to place the materials in context (not as successful as I hoped, though). I am usually frustrated with textbooks on this point, e.g., Section 1.1 of our text, which is supposed to explain why one should study automata. On another front, reading technical papers in this area tends to be too demanding for undergraduate students. I didn't do it two years ago. But this time, I felt that the two papers are indeed readable and beneficial for promoting your

critical attitude. I also tried to promote collaborating atmosphere; you did a good job on this. Although everyone of us has some competitiveness, excessive competitiveness can be very harmful (Rosenau).

Over the past few years, I have been revising the evaluation procedure, partly based on the ideas in the education literature (e.g., Huber & Freed, Fink). Whatever I do, there always are students who don't like it. But I cannot stop experimenting with new approaches/materials. My wife (also a college teacher) and I often wonder why so many college teachers (who did a PhD) don't apply their research attitude to teaching. Not every attempt would be successful, but no attempt must be self-limiting. In the end, I am glad that I did this course.

One other thing, which is personal, is that I got a Da.D. on March 14. I tried to minimize certain adverse effects on this course, but I have to admit that it was the single most challenging event I experienced in my life (even more than a Ph.D.). But I hope you enjoyed some baby examples. As I mentioned, child-parent interaction is highly complex (Schore). Actually, there is a book on this topic written from a complex systems point of view (Siegel), which was a textbook for my section of First Year Seminar last semester (http://www.tcnj.edu/~komagata/fsp111/04f/). In case you plan to become a parent, I highly recommend this book; you may even find something new about the relationship with your parents.

Thank you for taking this course (with your *free will*) and contributing to the class *interactively*. I'd be happy if you learned something in this course *as a result of your own motivation* (not because you were required to do so). That way, you will continue to learn. Regardless of your career choice, I wish you the best! Please let me know what happens next in your life.

**Additional References** (only those not in the course on-line references)

[L] indicates that the resource is available in the TCNJ library


- Engel, Susan. 1999. *Context is everything: the nature of memory*. W.H. Freeman. [L]
- Fink, L. Dee. 2003. *Creating Significant Learning Experiences: An Integrated Approach to Designing College Courses*. Jossey-Bass.
- Huba, Mary E. and Freed, Jann E. 2000. *Learner-centered assessment on college campuses: shifting the focus from teaching to learning*. Allyn and Bacon. [L]
- Kennedy, James F., Eberhart, Russell C., and Shi, Yuhui. 2001. *Swarm intelligence*. Morgan Kaufmann Publishers. [L]
- LeDoux, Joseph. 1999. Emotion, Memory, and the Brain. In *the Scientific American book of the brain*. Lyons Press.
- Rosenau, Pauline Vaillancourt. 2003. *The competition paradigm: America's romance with conflict, contest, and commerce*. Rowman & Littlefield Publishers.
- Schore, Allan N. 1994. *Affect regulation and the origin of the self: the neurobiology of emotional development.* Lawrence Erlbaum Associates. [L]
- Siegel, Daniel J. 1999. *The developing mind: toward a neurobiology of interpersonal experience*. Guilford Press. [L]


// End

**The Notion of “Theory” in Formal Logic**
Nobo Komagata, January 14, 2005

While the notion of “theory” is used in all sorts of context, we rarely agree on the term completely. As a background discussion to understand the notion better, these notes explores the idea developed in formal logic. The materials are taken from standard logic textbooks [Enderton, 2001; Ebbinghaus et al., 1984] and a textbook on program specification Loeckx et al. [1996].

# 1 Logic

The presentation of logic in this subsection is intentionally left rather vague to abstract away from the details. A formal presentation based on first-order logic and many-sorted signature are found in the texts.

Informally, a logic is a specification of a language and its meaning. We first introduce the components of logic.

**Definition 1** (**Language**) A language is a specification of the use of symbols.

Depending on further specification, these symbols are combined into a term, a formula, or a sentence. For example, a language of first-order logic may have a term $wife(Socrates)$, a formula $man(x) \to mortal(x)$, and a sentence $\forall x\,[man(x) \to mortal(x)]$. A language of equational logic may have a formula $x+1=1+x$. Although it is not technically correct (e.g., Enderton [2001]), we call the expressions we deal in these notes (in correspondence to “statement”) as “formula.”[1]

**Definition 2** (**Interpretation**) Interpretation is a function (mapping) that specifies the meaning of symbols *built in* the logic under consideration.

For example, a logic of arithmetic may have a special constant ‘0’, whose meaning is fixed in the logic (e.g., the integer ‘0’ as we understand). Many logics also have logical connectives such as ‘$\wedge$’ (and), whose interpretation may roughly correspond to that of natural language “and.”[2]

**Definition 3** (**Structure**) A structure (for a logic) is a function (mapping) that defines the meaning of symbols not defined by the interpretation.

By convention, for each sort (type) of elements, a structure assigns the universe of individuals corresponding to that sort. For each operation (in first-order logic, function or relation), a structure assigns its meaning in terms of the involved individuals. For example, a first-order logic assigns to the universal quantifier ‘$\forall$’ a set of integers (individuals). The same first-order logic may have a relation “even” that assigns “true” for any even numbers. For a certain set of algebraic symbols, we may consider a structure of an algebra, e.g., group.

**Definition 4** (**Satisfaction**) A satisfaction is a relation between structures and formulas. For a structure $A$ and a formula $\varphi$, this relation is usually written as $A \vDash \varphi$.

For example, a structure of group satisfies a formula “$x+(y+z)=(x+y)+z$” (associativity).

We are now in a position to define logic in terms of the above-mentioned components.

**Definition 5** (**Logic**) A logic is a combined specifications of language, interpretation, and satisfaction.

The language of a logic is also called syntax. By fixing the syntax, we can tell whether a certain expression is a formula of that logic. The remaining elements define the semantics of the logic. By fixing the semantics, we can tell whether a certain formula is satisfied by a certain structure.[3]

[1] In first-order logic, the distinction between “formulas” and “sentences” is that in the latter, every variable is “bound” by a quantifier. In equational logic, all variables in a “formula” are assumed to be universally quantified.

[2] For the interpretation of a formula with unbound variable, the “assignment” of variable also needs to be specified. Since we focus on quantified formulas, we skip the definition of assignment.

[3] Both the language and the structure of a logic may be specified in terms of a signature $\Sigma$, the specification of the involved “sorts” and “operations.” In this case, the language and the structure can be represented as $L(\Sigma)$ and $Alg(\Sigma)$, called $\Sigma$-algebra. While the traditional first-order logic deals with a single “sort,” i.e., sort of individuals, modern programming specification often uses multiple “sorts,” corresponding to multiple (often complex) data types.

## 2 Definition of "Theory"

We now proceed to the definition of "theory" in the context of formal logic. We need to introduce a few more definitions about the relation between structures and formulas.

**Definition 6** (**Model**) For a logic $L$, a structure $A$ is called a model of formulas $\Phi$, if $A \vDash \varphi$ for all $\varphi \in \Phi$.

By abusing the symbol '$\vDash$', we also write $A \vDash \Phi$. For example, a structure (algebra) of group is a model of formulas $\{x+(y+z)=(x+y)+z\}$. The set of all models of $\Phi$ is written as $Mod(\Phi)$.

**Definition 7** (**Logical consequence**) For a logic $L$, a formula $\varphi$ is called a logical consequence of $\Phi$, if for each $A \in Mod(\Phi)$, $A \vDash \varphi$.

By still abusing the symbol '$\vDash$', we also write $\Phi \vDash \varphi$. For example, "$x+(y+z)=(x+y)+z$" is a logical consequence of the axioms of group.

Let us now proceed to define the bare-bone definition of "theory." Informally, a theory is a set of formulas that is closed under logical consequence, or complete with respect to validity.

**Definition 8** (**Theory**) For a logic $L$, a theory is a set of formulas $\Phi$ such that for each formula $\varphi$, $\Phi \vDash \varphi$ implies $\varphi \in \Phi$.

In other words, if we can derive from a set of formulas a formula that does not belong to the set, the set is not a theory. At this point, the only requirement is that the set of formulas is validly closed. It does not even need to be consistent. Thus, an inconsistent set of formulas is still a theory as long as it satisfies the above definition. This definition of "theory" is good only if there is a mechanism to represent such a closed set. In some cases, we may list all the formulas extensionally. But it is not usually the way an interesting theory is described.

A more concise representation of a theory is based on the idea of axiomatization. Let us proceed with a few more definitions necessary for this move.

**Definition 9** (**Theory of structures**) For a logic $L$, the theory of a class of structures $\mathcal{C}$ is the set: $Th(\mathcal{C}) = \{\varphi \mid \mathcal{C} \vDash \varphi\}$.

**Definition 10** (**Consequences of formulas**) $Cn(\Phi) = Th(Mod(\Phi))$.

**Definition 11** (**Axiomatizable theory**) A theory $\Phi$ is axiomatizable iff there is a decidable set $\Psi$ such that $\Phi = Cn(\Psi)$.

A theory may or may not be axiomatizable. In addition, the set of axioms may be finite or infinite. Naturally, many interesting cases of axiomatization is finite. For example, the theory of group is finitely axiomatizable (in a certain logic). This point is particularly important because finite axiomatization allows us to *predict* beyond what has already been described.

## 3 Axiomatization via Calculus

Although the above presentation provides a view about theory sufficient as a background, it is often inconvenient or impossible to deal with models in a direct way. Fortunately, the study of logic can provide useful results on the relation between semantic analysis via models and syntactic (mechanical) analysis via formulas.

Let us first introduce the notion of "calculus." Informally, a calculus is a purely syntactic manipulation of symbols to derive a new formula from a set of formulas.

**Definition 12** (**Axioms**) We consider a finite set of axiom schemata, each of which consists of a decidable subset of the formulas of $L$.

Here we talk about axiom schemata rather than axiom instances. This is convenient because we can often consolidate infinitely many axioms as a single axiom scheme. For simplicity, we may call axiom schemata as axiom. We have seen an axiom of group theory.

**Definition 13** (**Rules of inference**) A rule of inference is a finite, decidable relation between a set of formulas and a formula.

A typical rule of inference is 'modus ponens', i.e., from "if $p$, then $q$" and "$p$", to derive "$q$".

**Definition 14** (**Calculus**) A calculus is a combined specification of axioms and rules of inference.

**Definition 15** (**Derivation**) Derivation of a formula $\varphi$ from a set of formulas $\Phi$ in a calculus is specified by axioms and rules of inference, and is written as $\Phi \vdash \varphi$.

**Definition 16** (**Soundness**) A calculus for a logic is sound if $\Phi \vdash \varphi$ implies $\Phi \models \varphi$.

**Definition 17** (**Completeness**) A calculus for a logic is complete if $\Phi \models \varphi$ implies $\Phi \vdash \varphi$.

A sound and complete calculus of a logic guarantees that mechanical operations in the calculus is semantically valid. In practice, it is often considered sufficient to use a sound calculus (i.e., mechanical operations are always correct, but there may not be a proof for some true formula).

Now, let us return to the topic of axiomatization. An axiomatizable theory can be completely specified by a sound and complete calculus. A sound calculus can describe an axiomatizable theory, but the specification may not be complete. Thus, the ability of a mechanical specification (axiomatization) of a theory via calculus depends on the properties associated with the calculus.

## Bibliography

Heinz-Dieter Ebbinghaus, Jèorg Flum, and Wolfgang Thomas. 1984. *Mathematical logic*. Undergraduate texts in mathematics. Springer-Verlag, New York.

Herbert B. Enderton. 2001. *A mathematical introduction to logic*. Harcourt, San Diego, CA, 2nd edition.

Jacques Loeckx, Hans-Dieter Ehrich, and Markus Wolf. 1996. *Specification of abstract data types*. Wiley, Chichester, UK.